TUGAS AKHIR – TE 141599

PENGEMBANGAN SISTEM PENDARATAN OTOMATIS PADA PESAWAT TANPA AWAK

Erwan Aprilian
NRP 2214 105012

Dosen Pembimbing
Ronny Mardiyanto, ST.,MT.,Ph.D.

JURUSAN TEKNIK ELEKTRO
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya 2017

FINAL PROJECT – TE 141599

DEVELOPMENT OF AUTOLANDING SISTEM FOR UNMANNED AERIAL VEHICLE (UAV)

Erwan Aprilian
NRP 2214 105012

Supervisors
Ronny Mardiyanto,ST.,MT.,Ph.D.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Industrial Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan tugas akhir saya dengan judul **"Pengembangan Sistem Pendaratan Otomatis pada Pesawat Tanpa Awak"** adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Januari 2017

Erwan Aprilian
NRP. 2214105012

PENGEMBANGAN SISTEM PENDARATAN OTOMATIS PADA PESAWAT TANPA AWAK

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Jurusan Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui

Dosen Pembimbing ,

Ronny Mardiyanto, ST., MT., Ph.D.
NIP. 198101182003121003

SURABAYA
JANUARI, 2017

PENGEMBANGAN SISTEM PENDARATAN OTOMATIS PADA PESAWAT TANPA AWAK

Nama : Erwan Aprilian
Dosen Pembimbing : Ronny Mardiyanto, ST.,MT.,Ph.D.

ABSTRAK

UAV adalah salah satu wahana tanpa awak di udara yang dapat terbang tanpa pilot. Secara teknis kemampuan terbang dan sistem kerja pesawat ini menyerupai kondisi pesawat sebenarnya. Selanjutnya UAV tersebut dapat digunakan untuk keperluan baik dilingkup militer maupun sipil. Pada proposal tugas akhir ini akan dirancang dan direalisasikan pengembangan sistem *autopilot* pada UAV dengan tambahan mendarat otomatis. Sistem ini menggunakan kontrol manual dan *autopilot*. Pada mode manual, pengguna secara manual mengendalikan pergerakan pesawat melalui RC sedangkan pada mode *autopilot* pesawat dikendalikan oleh mikrokontroler yang mengolah data-data sensor untuk mengarahkan pesawat dari satu titik ke titik yang lain. Sensor yang digunakan antara lain IMU (*Inertial Measurement Unit* yang didalamnya terdapat *gyroscope* dan *accelerometer*), GPS, *compass, barometric altimeter, airspeed* dan *ultrasonic*. Mikrokontroler menerima data input dari sensor secara terus-menerus, mengolah sampel data dan menghasilkan output untuk menggerakkan aktuator berupa servo. Pengolahan data sensor menggunakan kontrol PID (*Proportional Integral Derivative*) dengan metode Ziegler Nichols. Manuver kontrol *autopilot* meliputi *roll* (sumbu x), *yaw* (sumbu y), *pitch* (sumbu z), ketinggian dan *heading* pesawat. Pesawat akan terkoneksi dengan *ground station* melalui perangkat telemetri. Sistem navigasi ini dapat secara tepat mengarahkan pesawat menuju satu titik atau lebih dengan toleransi kesalahan ≤ 10 meter pada ketinggian 90-100 meter. Selain itu pesawat dapat terbang dengan radius ± 2 km dari *ground station*.

Kata kunci – UAV, IMU, PID

[Halaman ini sengaja dikosongkan]

DEVELOPMENT OF AUTOLANDING SISTEM FOR UNMANNED AERIAL VEHICLE (UAV)

Name *:* Erwan Aprilian
Supervisor I : Ronny Mardiyanto,ST.,MT.,Ph.D.

ABSTRACT

UAV is a unmanned air vehicle that can fly without a pilot. Technically the ability to fly and work systems of these aircraft resemble the actual condition of the aircraft. Furthermore, the UAV can be used for both military and civilian. At the proposal of this thesis will be designed and realized the development of the UAV autopilot system with the addition of an automatic landing. This system uses manual control and autopilot. In manual mode, the user manually controls the movement of air through the RC aircraft while on autopilot mode is controlled by a microcontroller that processes sensor data to direct air from one point to another. Sensors are used among others IMU (Inertial Measurement Unit which there gyroscope and accelerometer), GPS, barometric altimeter, airspeed and ultrasonic. Microcontroller receives input data from the sensors continuously, process data samples and generate output to drive the servo actuator form. Processing of sensor data using PID control (Proportional Integral Derivative) with Ziegler Nichols method. Maneuver autopilot includes roll control (x-axis), yaw (y-axis), pitch (the z axis), aircraft altitude and heading. The aircraft will be connected with the ground station via telemetry devices. The navigation system is expected to accurately direct the plane to a point or more at fault tolerance ≤ 10 meters at an altitude of 90-100 meters. Besides the aircraft can fly with a radius of ± 2 km from the ground station.

Keywords *– UAV, IMU, PID*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Alhamdullilah, puji syukur penulis ucapkan kehadirat Illahi Rabbi, Allah SWT atas segenap karunia-Nya yang tak terhitung jumlahnya, sehingga penulis dapat menyelesaikan penulisan buku Tugas Akhir dengan judul **"Pengembangan Sistem Pendaratan Otomatis pada Pesawat Tanpa Awak"**. Tugas akhir merupakan salah satu syarat yang harus dipenuhi untuk menyelesaikan program studi Strata-1 pada Jurusan Teknik Elektro Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember Surabaya. Penulis menyadari bahwa tugas akhir ini adalah bagian dari proses pematangan kapasitas akademik dalam banyak hal.

Pada kesempatan ini penulis ingin menyampaikan ucapan terimakasih dan penghargaan setingi-tingginya kepada :

1. Keluarga penulis ayahanda Herwanto, ibunda Warlika, dan istri Angger Martalia beserta seluruh keluarga yang selalu memberikan doa, motivasi, semangat, perhatian dan kasih sayangnya.
2. Bapak Ronny Mardiyanto, ST., MT., Ph.D., selaku dosen pembimbing serta bapak Heri Suryo Atmojo, ST., MT., Ph.D yang selalu memberikan arahan dan ilmu dalam penyelesaian Tugas Akhir ini.
3. Seluruh dosen bidang studi Elektronika Jurusan Teknik Elektro FTI ITS.
4. Rahmad Hidayat, Hardianto dan teman-teman Teknik Elektro, serta banyak pihak yang tidak dapat disebutkan satu-persatu.
5. Komunitas *Magic Aeromodelling Club* Surabaya atas ilmu dan pengalaman yang penulis dapatkan.

Penulis menyadari bahwa dalam tugas akhir ini masih belum sempurna. Semoga ketidak sempurnaan itu menjadi celah untuk proses belajar dan pengembangan rancangan kedepannya. Semoga Tugas Akhir ini bisa menjadi bahan referensi dan memberikan manfaat bagi pengembangan dunia elektonika.

Surabaya, Januari 2017

Penulis

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

[Halaman ini sengaja dikosongkan]

BAB 1
PENDAHULUAN

1.1 Latar Belakang

UAV adalah salah satu wahana nirawak di udara. Pergerakan UAV secara manual di udara dapat dikendalikan menggunakan *radio controller*. Ruang udara yang luas dan bebas rintangan memberikan kebebasan dalam mengendalikan pesawat ini. Manuver pesawat seperti *rolling*, *pitching* dan *yawing* membuat pengguna seakan mengendalikan pesawat sebenarnya. Ketinggian, kecepatan, daya jangkau, fleksibilitas dan mobilitas pesawat di udara menuntut kemampuan khusus pengguna untuk mengendalikan pesawat secara tepat dan akurat.

Kekurangan dari UAV yang sudah ada yaitu mahalnya biaya pengadaannya. Selain itu sistem autopilot yang sudah ada adalah produk dari luar negeri sehingga kita kedepannya akan mengalami ketergantungan teknologi. Dengan biaya operasional yang murah dan resiko yang jauh lebih kecil dari pada pesawat berawak mendorong penulis untuk melaksanakan perancangan UAV. Melalui penembangan sistem pendaratan otomatis pada pesawat tanpa awak UAV dengan GPS *waypoint* ini, penulis berharap bahwa bangsa Indonesia tidak mampu mengembangkan teknologi kedirgantaraan dengan asas kemandirian sehingga teknologi tersebut dapat digunakan sebagai solusi alternatif untuk mengatasi berbagai kendala di lapangan secara efektif dan efisien.

Pesawat *fix-wing* RC aeromodeling sebagai ide dasar UAV saat ini umumnya menggunakan RC dengan frekuensi *hopping* 2,4 GHz. Aktuator pada pesawat menggunakan motor DC *brushless* dan motor servo. *Propeller* dipasang pada motor *brushless* sebagai tenaga penggerak pesawat. Motor servo terhubung dengan *aileron* (*roll*), *elevator* (*pitch*) dan *rudder* (*yaw*) sebagai kendali kemudi pesawat. Catu daya elektrik pesawat menggunakan baterai untuk mensuplai komponen elektrik dan ESC (*Electronic Speed Conroller*) untuk mengatur aliran arus motor *brushless* dan sebagai *source power* untuk *mikrokontroller* beserta perangkat sensor.

Sistem kendali UAV yang akan dirancang pada tugas akhir ini menggunakan sistem otomatis. Secara manual pergerakan pesawat dikendalikan pengguna melalui RC. *Receiver* pada pesawat akan menerima perintah dari pengguna dan diaktualisasikan dengan bekerjanya tenaga penggerak dan kendali kemudi pesawat. *Autopilot*

digunakan pada saat pesawat sudah mengudara dengan ketinggian tertentu. Pesawat dilengkapi dengan sensor IMU, *barometric altitude* dan magnetometer. Sensor-sensor tersebut memberikan informasi *motion*, ketinggian dan *heading* pesawat. Selain itu pesawat menggunakan GPS untuk mengetahui lokasi dan koordinat dari pesawat, sensor *airspeed* untuk mengetahui kecepatan pesawat dan sensor *ultrasonic* untuk mengontrol ketinggian real jarak dekat.

Pesawat secara *autopilot* akan mengudara dari satu titik ke titik lain dengan koordinat yang ditentukan oleh GPS. Data dari sensor diolah oleh kontroler untuk menggerakkan aktuator sehingga pesawat akan menuju koordinat yang ditentukan. Pergerakan pesawat menggunakan kontrol PID dengan metode Ziegler Nichols. Data informasi sensor dan perintah kendali *autopilot* akan ditampilkan melalui komputer sebagai *ground station* menggunakan perangkat telemetri.

1.2 Rumusan Masalah

Sehubungan dengan latar belakang yang telah dijelaskan sebelumnya, terdapat beberapa masalah yang akan dibahas antara lain sebagai berikut :

1. Bagaimana mengendalikan UAV dengan sistem kontrol manual.
2. Bagaimana merancang sebuah UAV sehingga mampu terbang dengan sistem kestabilan di udara.
3. Bagaimana merancang dan mengimplementasikan sistem navigasi otomatis dengan GPS *waypoint* dari satu titik titik koordinat ke koordinat lain untuk pendaratan otomatis.

1.3 Batasan Masalah

Batasan masalah yang akan dibahas dalam Tugas Akhir ini adalah :

1. Mengendalikan UAV dengan sistem kontrol manual.
2. Perancangan kontroler otomatis pada UAV dengan GPS *waypoint* untuk pendaratan.
3. Implementasi sensor IMU (*gyroscope* dan *accelerometer*) pada kontroler otomatis untuk kestabilan pesawat di udara menggunakan *turnigy T1000FC* dan sistem navigasi menggunakan GPS.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah merancang dan merealisasikan pengembangan sistem navigasi otomatis pada UAV dengan GPS *waypoint* untuk pendaratan otomatis, dimana mikrokontroler berfungsi sebagai kontroler *autopilot*. Pergerakan pesawat dan data-data penerbangan dapat dipantau melalui komputer.

1.5 Metodologi

Metodologi yang digunakan dalam penyusunan tugas akhir ini sebagai berikut :

1. Studi literatur
   Studi literatur tugas akhir ini bersumber pada jurnal-jurnal, buku referensi dan *datasheet* komponen yang digunakan.
2. Perancangan dan realisasi alat
   Perancangan alat pada tugas akhir ini dimulai dari perancangan dan pembuatan pesawat, sistem kontrol manual pada pesawat, pembacaan sensor IMU, *barometric altitude* dan penggunaan koordinat GPS, perancangan sistem *autopilot* dengan mikrokontroler dan mendesain kontroler PID dengan metode Ziegler Nichols.
3. Pengujian alat
   Pengujian alat dimulai dengan terlebih dahulu menguji respon pada tiap-tiap sensor secara parsial . Apabila respon pada tiap sensor sudah sesuai, maka dilakukan penggabungan beberapa sensor menjadi sebuah sistem dan dilanjutkan dengan uji fungsi sistem.
4. Penulisan laporan
   Laporan yang ditulis akan mengacu pada perancangan dan realisasi alat, serta hasil dari pengujian alat.

1.6 Sistematika Penulisan

Sistematika penulisan pada tugas akhir ini dibagi menjadi beberapa bab dengan rincian :

BAB 1 : PENDAHULUAN
Menguraikan latar belakang, perumusan masalah, batasan masalah, tujuan dan manfaat yang berkaitan dengan pengerjaan dan penyusunan Tugas Akhir ini.

BAB 2 : TEORI PENUNJANG

Pada bab ini dikemukakan berbagai macam dasar teori yang berhubungan dengan permasalahan yang dibahas, antara lain meliputi teori tentang pengenalan UAV, mikrokontroler, sensor *accelerometer* dan *gyroscope, barrometric altimeter*, GPS, *airspeed*, *ultrasonic* dan kontrol PI.

BAB 3 : PERANCANGAN DAN REALISASI ALAT

Berisi tentang tahap-tahap perancangan sistem kontrol manual dan otomatis yang terpasang pada pesawat, baik secara *hardware* dan *software*, kontroler PID dan penempatan kontroler di pesawat.

BAB 4 : PENGUJIAN DAN ANALISA SISTEM

Bab ini membahas mengenai pengujian dari sistem yang telah diimplementasikan pada UAV dan analisa data berdasarkan parameter yang ditetapkan.

BAB 5 : PENUTUP

Berisi tentang kesimpulan dan saran yang diperoleh dalam Tugas Akhir ini.

1.7 Relevansi

Manfaat dari tugas akhir ini adalah mengetahui kerja sistem kontrol UAV baik secara manual maupun otomatis yang dilengkapi sensor *gyroscope- accelerometer, barrometric altimeter, airspeed, ultrasonic dan GPS.* Hasil yang dicapai diharapkan dapat menjadi salah satu referensi dalam pengembangan sistem autopilot wahana nirawak.

BAB 2
TEORI PENUNJANG

2.1 *Unmanned Aerial Vehicle* (UAV)

UAV secara umum memiliki definisi sebagai pesawat yang dapat terbang secara otomatis tanpa dikendalikan oleh pilot. Secara spesifik UAV memiliki definisi sebagai pesawat udara yang mempunyai sistem kontrol otomatis baik dalam stabilitas sikap di udara maupun sistem navigasinya sehingga dapat melaksanakan misi-misi tertentu tanpa diawaki oleh seorang pilot di dalamnya. UAV juga sering disebut dengan nama Pesawat Udara Nir Awak (PUNA). Karena tidak memiliki awak, UAV harus dikendalikan dari jarak jauh menggunakan remote control dari luar kendaraan atau biasa disebut *Remotely Piloted Vehicle* (RPV). Selain itu, UAV juga dapat bergerak secara otomatis berdasarkan program yang sudah ditanamkan pada sistem mikrokontrolernya.

2.1.1 Jenis UAV

Secara umum pembagian jenis UAV dilakukan menurut jenis , sumber tenaga pengerak dan besar atau berat pesawat.

1. Jenis pesawat UAV berdasarkan jenis sayap.

   Jenis sayap UAV terbagi dalam 2 bagian yaitu:

   a. *Fix wing*. Pesawat model *fixwing* adalah pesawat yang memiliki bentuk sayap tetap atau tidak bergerak. Pesawat mendapatkan *thrust* dari gaya dorong motor yang menerpa bagian sayap yang memiliki bentuk airfoil tertentu dari depan sampai belakang sehingga menghasilkan gaya angkat.

   b. *Rotary wing*. Pesawat model *rotary wing* memiliki sayap yang bergerak atau berputar atau baling-baling sehingga menghasilkan gaya angkat. Pergerakan pesawat diatur melalui perubahan sudut serang posisi baling-baling.

Gambar 2.1 Jenis sayap UAV

2. Jenis pesawat berdasarkan sumber tenaga.
   a. *Combustion engine*. Sumber tenaga pesawat menggunakan pembakaran bahan bakar cair pada engine untuk menggerakkan propeler pesawat. Kelebihan pesawat jenis ini memiliki kecepatan tinggi, mengudara dalam waktu lama dengan daya jelajah jauh. Kekurangan dari pesawat ini adalah getaran dan suara keras dari pembakaran *engine*.
   b. *Elcktric*. Sumber tenaga pesawat menggunakan suplai daya dari baterai untuk menggerakkan propeler pesawat. Kekurangan pesawat jenis ini memiliki kecepatan standar, mengudara dalam waktu relatif singkat dengan daya jelajah menengah. Kelebihan pesawat jenis ini adalah getaran dan suara yang lebih halus.

Gambar 2.2 Tenaga penggerak UAV

3. Jenis UAV berdasarkan berat.

   a. UAV *super heavy*

   UAV super heavy adalah jenis UAV yang memiliki berat diatas 2000 Kg. Sebagai contoh UAV *super heavy* adalah *Global Hawk* seperti terlihat pada gambar berikut.

Gambar 2.3 UAV *Super Heavy Global Hawk*

b. UAV *heavy*

UAV *heavy* adalah jenis robot penjelajah udara dengan berat antara 200 – 2000 Kg. Salah satu contoh UAV *heavy* adalah A-160.

Gambar 2.4 UAV A-160

c. UAV *medium*

UAV *medium* adalah robot penjelajah udara yang memiliki berat pada range 50-200Kg. Contoh UAV jenis *medium* adalah UAV *Chyper*.

Gambar 2.5 UAV *Chyper*

d. UAV *light*

UAV *light* merupakan robot penjelajah udara dengan bobot 5-50Kg. Contoh UAV jenis *light* adalah UAV *Neptune.*

Gambar 2.6 UAV *Neptune*

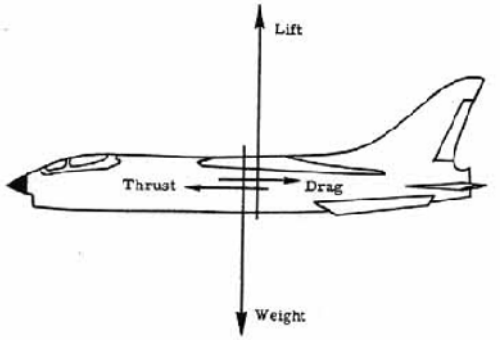
e. UAV *Micro*

UAV *micro* adalah robot penjelajah udara yang ringan dan memiliki bobot kurang dari 5kg. Contoh UAV *micro* adalah UAV *Dragon Eye.*

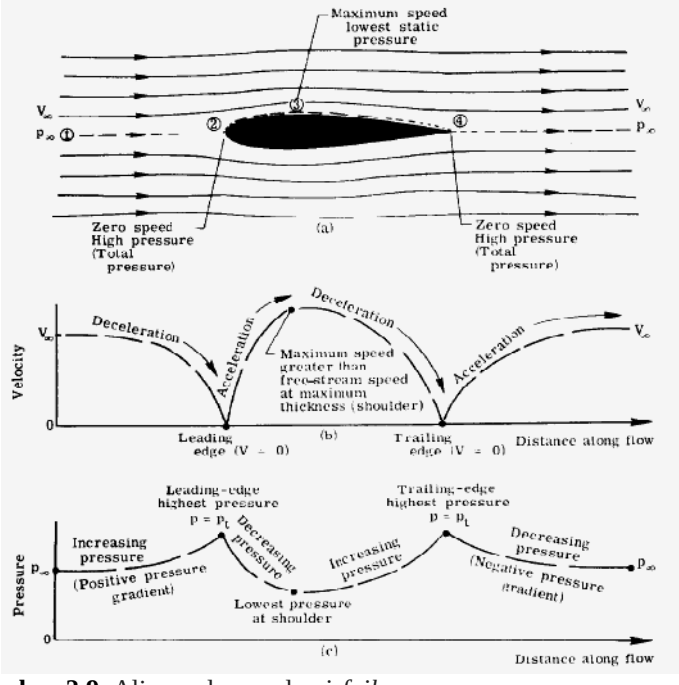
Gambar 2.7 UAV *Dragon Eye*

2.1.2 Teori Pesawat Terbang

Pada pesawat terbang bekerja empat buah gaya yaitu gaya dorong(*thrust*), gaya hambat(*drag*), gaya angkat(*lift*) dan gaya gravitasi bumi karena berat pesawat(*weight*). Gaya dorong muncul akibat dorongan angin dari baling-baling yang digerakkan oleh motor atau mesin pesawat. Gaya hambat muncul karena luas penampang pesawat yang dihamtam oleh angin dari depan pesawat yang mengakibatkan pesawat terhambat untuk bergerak ke depan. Gaya angkat muncul karena tekanan angin atau udara pada bagian atas sayap lebih rendah dari bagian bawah pesawat. Gaya-gaya yang bekerja pada dapat dilihat pada gambar 2.8.



Gambar 2.8 Gaya pada Pesawat

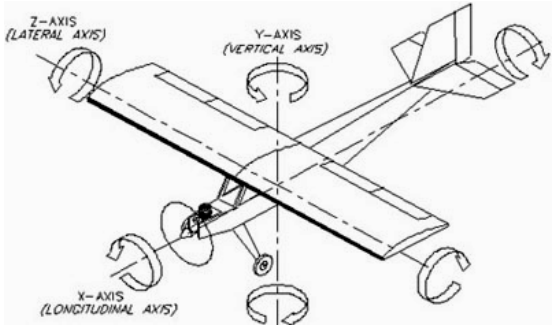


Gambar 2.9 Aliran udara pada *airfoil*

Angin yang menerpa bentuk *airfoil* sayap menyebabkan kecepatan angin pada bagian atas sayap lebih tinggi dari pada bagian bawah sayap, akibatnya tekanan pada bagian atas sayap lebih rendah dari pada bagian bawah sayap. Hal tersebut menyebabkan pesawat mendapat gaya angkat. Proses terjadinya gaya angkat dapat dilihat pada gambar 2.9.

Selain empat macam gaya yang terjadi pada pesawat, ada tiga sumbu gerak meliputi sumbu longitudinal, vertikal dan lateral. Titik pertemuan sumbu-sumbu tersebut merupakan CG (*Center of Gravity*) dalam kestabilan dan manuver pesawat. Manuver pesawat antara lain:

a. *Roll*, gerak pesawat terhadap sumbu longitudinal dengan menggunakan aileron.
b. *Pitch*, gerak pesawat terhadap sumbu lateral dengan menggunakan elevator.
c. *Yaw*, gerak pesawat terhadap sumbu vertikal dengan menggunakan rudder.



Gambar 2.10 Sumbu gerak pesawat

2.1.3 Radio Kontrol

Radio kontrol adalah perangkat elektronika yang digunakan untuk mengatur pergerakan pesawat UAV. Modul radio kontrol terdiri dari sistem *transmitter* dan *receiver*. *Transmitter* mengirimkan sinyal kontrol menuju *receiver* melalui *channel* yang dimiliki secara *wireless*.

Joystik pada *transmitter* mewakili perubahan sinyal yang akan dipancarkan. *Receiver* yang terpasang pada badan pesawat UAV akan menangkap sinyal kontrol dan diimplementasikan oleh motor *brushless* dan servo di pesawat. Radio kontrol yang digunakan adalah Fly Sky FS-i6 yang mempunyai 6 *channel*.

Gambar 2.11 Radio kontrol tipe Fly Sky FS-i6

Tabel 2.1 Spesifikasi radio kontrol Fly Sky FS-i6

Fitur	Keterangan
Channel	6 channel
Sinyal data	AFHDS 2A 2.4GHz
Sistem program	Komputerisasi, layar sentuh
Memori program	<=20dbm
Catudaya	4.0V~6.5V

2.1.4 Perangkat Elektromekanik pada UAV

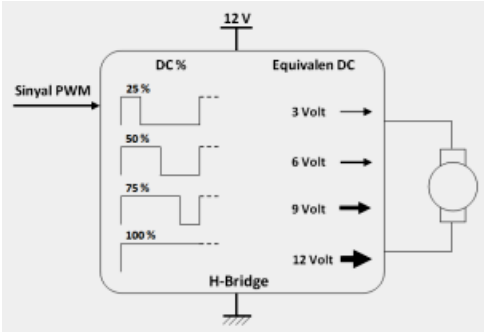
UAV yang digunakan dalam tugas akhir ini menggunakan sumber tenaga elektrik. Perangkat elektromekanik yang terdapat pada UAV ini antara lain sebagai berikut:

a. Motor DC Brushless

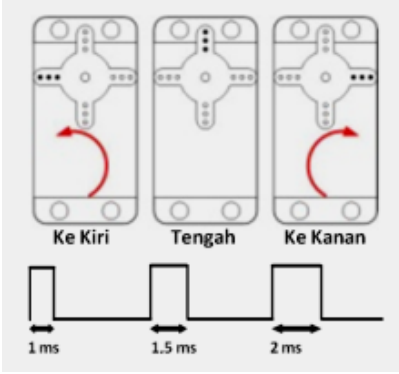
Brushless motor DC adalah motor DC elektrik dengan eksitasi terpisah, yang aktif apabila mendapat suplai tegangan searah. Tegangan DC input akan dikonversi menjadi sinyal AC untuk menggerakkan motor. Sinyal kontrol kecepatan pergerakan motor dihasilkan dari *receiver*.

b. ESC (*Electronic Speed Controller*)

Adalah perangkat elektronika yang digunakan untuk mengatur kecepatan putar motor *brushless*. ESC menerima input pulsa dari *receiver* dan mengkonversikan pulsa tersebut ke dalam bentuk pengaturan daya yang akan disuplai dari catu daya ke motor DC *brushless*.



Gambar 2.12 Konversi PWM pengatur kecepatan motor DC *brushless*



Gambar 2.13 Pengaturan arah servo dengan PWM

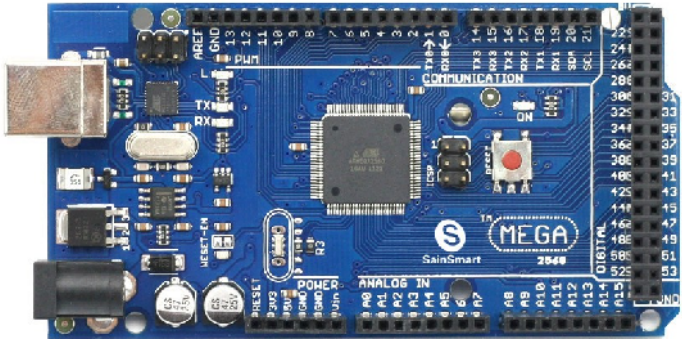
ESC menggunakan transistor power MOSFET yang tersusun dalam rangkaian H-*Bridge*. Umumnya menggunakan kapasitor tambahan untuk melindungi ESC dari perubahan nilai pengaturan daya yang bervariatif .

c. Motor Servo

Motor servo adalah motor elektrik yang menggunakan suplai tegangan searah. Servo membutuhkan input kontrol berupa PWM dengan frekuensi 50 Hz untuk mengubah tegangan pada potensiometer, agar dapat memutar motor. Lebar pulsa kontrol bervariasi tergantung dari spesifikasi servo yang digunakan. Umumnya servo berada pada posisi tengah apabila diberi pulsa kontrol dengan lebar 1500 µs.

2.2 Mikrokontroler

Mikrokontroler berfungsi untuk memproses dan mengolah input data dari *receiver* dan sensor-sensor untuk menghasilkan output penggerak aktuator. Mikrokontroler yang digunakan adalah Arduino Mega 2560. Arduino Mega 2560 adalah papan mikrokontroler berbasiskan ATmega2560. Arduino Mega 2560 memiliki 54 pin digital *input/output*, dimana 15 pin dapat digunakan sebagai *output* PWM, 16 pin sebagai input analog, dan 4 pin sebagai UART (*Universal Asynchronous Receiver Transmitter*), 16 MHz kristal osilator, koneksi USB, *jack power, header* ICSP, dan tombol reset.



Gambar 2.14 Arsitekktur ATmega 2560

ATMega 2560 sendiri memiliki 11 *port* (*port* A,B,C,D,E,F,G,H,J,K,L) dan memiliki jumlah total pin sebanyak 100 pin. Chip tersebut memiliki pin SDA (*Serial Data*) dan SCL (*serial Clock*), 4 komunikasi serial dan *pin change interupt* yang sangat memungkinkan jika digunakan untuk keperluan sistem kontrol yang banyak menggunakan sensor dan jalur komunikasi serial. Kelebihan arsitektur ATMega 2560 dijadikan pertimbangan oleh penulis dalam melaksanakan perancangan Tugas Akhir ini.

2.3 Turnigy T1000FC

Turnigy T1000FC adalah suatu perangkat *autopilot* yang berfungsi untuk 5 *mode* penerbangan dapat dipilih di T1000FC, Modus autopilot (auto level dan tidak membiarkan pesawat miring melewati 45 derajat) , mode 3D untuk olahraga terbang , Flight mode (stabilisasi off) , menuju terus , ketinggian terus , dan GPS kembali ke rumah, perangkat ini menggunakan IMU sebagai berikut ADXL345

accelerometer, AGD8 2135 *LUSDI gyro*, mikrokontroler Atmel MEGA328P (16MHz crystal), dan GPS H-8123.

Gambar 2.15 Turnigy T1000FC

2.4 GY-521 (MPU6050)

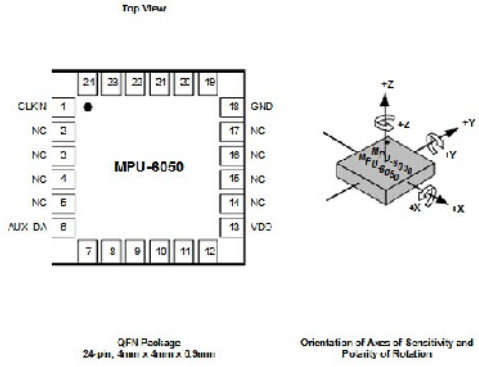
GY-521 MPU-6050 Module adalah sebuah modul berinti MPU-6050 yang merupakan 6 *axis Motion Processing Unit* dengan penambahan regulator tegangan dan beberapa komponen pelengkap lainnya yang membuat modul ini siap dipakai dengan tegangan supply sebesar 3-5VDC. Modul ini memiliki interface I2C yang dapat disambungkan langsung ke MCU yang memiliki fasilitas I2C.

Sensor MPU-6050 berisi sebuah MEMS *accelerometer* dan sebuah MEMS *gyro* yang saling terintegrasi. Sensor ini sangat akurat dengan fasilitas hardware internal 16 bit ADC untuk setiap kanalnya. Sensor ini akan menangkap nilai kanal *axis* X, Y dan Z bersamaan dalam satu waktu.

Berikut adalah spesifikasi dari Modul ini :

1. Berbasis Chip MPU-6050
2. Supply tegangan berkisar 3-5V
3. Gyroscope range + 250 500 1000 2000 ° / s
4. Acceleration range: ± 2 ± 4 ± 8 ± 16 g
5. Communication standard I2C
6. Chip built-in 16 bit AD converter, 16 bits data output

7. Jarak antar pin header 2.54 mm
8. Dimensi modul 20.3mm x 15.6mm



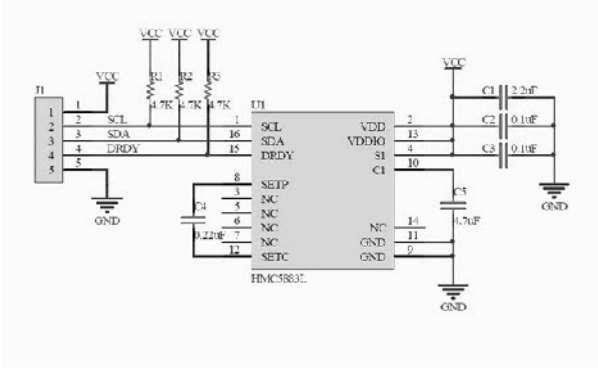
Gambar 2.16 Sumbu sensor GY-521

2.5 GY-273 (HMC5883L Compass)

Sensor kompas / magnetometer Triple Axis GY-273 ini menggunakan chip Honeywell HMC5883L yang merupakan chip untuk mendeteksi medan magnet lemah. Sensor ini dapat mengukur medan magnet antara 1.3 ~ 8 gauss untuk aplikasi kompas dan magnetometer. Menggunakan interface I2C, sensor ini mudah digunakan bersama Arduino atau Raspberry Pi, karena dapat bekerja di 3V hingga 5V.

Spesifikasi :

1. Supply Voltage : 3v ~ 5V
2. Protocol ; I2C
3. Range : 1.3 ~ 8 gauss
4. Automatic Degaussing Strap Drivers
5. Offset Cancelation
6. 12-bit ADC
7. Low power consumption (100uA)



8.

Gambar 2.17 Skematik HMC588L

2.6 Kestabilan Pesawat untuk pendaratan otomatis

Pesawat memiliki sudut pitch dan roll sensor *accelerometer* dan *gyroscope* dari *autopilot turnigy T1000FC* akan dikontrol oleh mikrokontroler arduino mega yang mengatur sensor *accelerometer* ,*compass* ,*barometer* ,GPS ,ultrasonic ,dan *airspeed*. Pada setiap pergerakan di udara, sistem akan merespon perubahan sudut pitch dan roll pesawat terhadap set poin (nilai error). Error kemudian dioleh oleh kontroler PI untuk menghasilkan nilai PWM penggerak servo *aileron* dan *elevator*. Pergerakan *aileron* dan *elevator* akan memberikan perubahan nilai sudut *pitch* dan *roll* pesawat ketika mengudara. Umpan balik *plant* ini berupa kedudukan pesawat setelah *aileron* dan *elevator* berubah.

2.6.1 Kontroler Proporsional

Kontroler proporsional adalah kontroler yang menghasilkan output kontrol sebanding dengan inputan. Persamaan dari kontroler proporsional adalah sebagai berikut:

$$u(t) = K_p(e(t) \tag{2.1}$$

2.6.2 Kontroler Integral

Kontroler integral adalah kontroler yang menghasilkan output kontrol berbanding lurus dengan besar dan lamanya eror. Integral dalam kontroler PID adalah jumlahan eror setiap waktu dan mengakumulasi *offset* yang sebelumnya telah dikoreksi. *Error*

terakumulasi dikalikan dengan *gain* integral (Ki) dan menjadi keluaran kontroler. Rumus kontroler integral adalah sebagai berikut :

$$I_{out} = K_i \int_0^t e(\tau)\, d\tau \tag{2.2}$$

Kontrol integral mempercepat perpindahan proses menuju *setpoint* dan menghilangkan *steady-state error* yang muncul pada kontroler proporsional. Namun, karena integral merespon terhadap eror terakumulasi dari sebelumnya, maka dapat menyebabkan *overshoot.*

2.6.3 Kontroler Derivatif

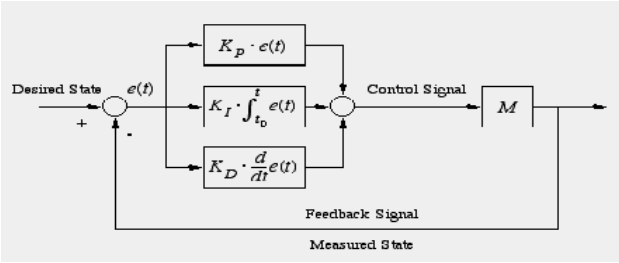
Kontroler derivatif adalah kontroler yang menghasilkan output kontrol sebanding dengan nilai derivatif dari inputan. Kontroler derivatif digabungkan dengan kontroler proporsional untuk meredam nilai *overshoot* dari output kontrol. Persamaan dari kontroler derivatif adalah sebagai berikut:

$$u(t) = K_p \tau_d (\frac{de(t)}{dt}) \tag{2.3}$$

2.6.4 Kontroler Proporsional Integral Derivatif

Kontroler proporsional integral derivatif adalah suatu sistem kontrol yang merupakan gabungan dari kontroler proporsional, kontroler integral dan kontroler derivatif dimana kontroler proporsinal akan menghasilkan output kontrol yang sebanding dengan nilai eror sehingga diperoleh nilai steady state, sedangkan kontrol integral berfungsi untuk meminimalisir nilai error steady state terhadap setpoint. Untuk mengurangi osilasi atau overshoot respon maka ditambahkan kontroler derivatif yang merupakan fungsi derivatif dari nilai eror dikalikan dengan konstanta derivatif sehingga persamaan kontrol PID adalah :

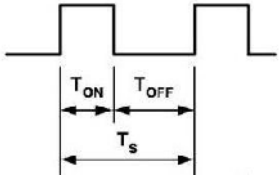
$$\mathrm{u}(t) = \mathrm{P}(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{2.4}$$



Gambar 2.18 Kontrol proporsional integral derivatif

2.7 *Pulse Width Modulation* (PWM)

PWM adalah sebuah metode manipulasi signal dari signal digital dimana variabel yang berubah adalah lebar pulsa *high* terhadap amplitudo dan frekuensi signal yang tetap. Sinyal PWM memiliki amplitudo dan frekuensi dasar yang tetap namun memiliki lebar pulsa yang bervariasi. Lebar pulsa PWM berbanding lurus dengan amplitudo sinyal asli yang belum termodulasi. Artinya, sinyal PWM memiliki frekuensi gelombang yang tetap namun memiliki *duty cycle* bervariasi antara 0% hingga 100%. Signal PWM dapat dikonversi menjadi nilai tegangan analog berdasarkan rumus sebagai berikut :



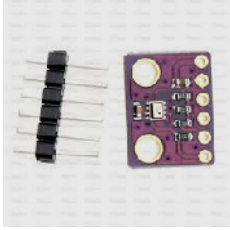
$$D = \frac{Ton}{Ttotal} \; x \; 100\% \qquad (2.5)$$

$$V\,out = D \; x \; Vin \qquad (2.6)$$

Gambar 2.19 Sinyal PWM digital

2.8 Sensor Barometrik Altimeter BMP280

BMP280 adalah sensor tekanan barometrik dengan range 300 s/d 1100 hPa sampai dengan 2.5hPa (1 hPa = 1 mbar) atau setara dengan +9000 sampai dengan -500 meter *above/below sea level*. Sensor ini memiliki tingkat ketelitian hingga ±0.12 hPa (equiv. to ±1 m). Ketinggian di permukaan bumi berhubungan dengan tekanan udara. Semakin tinggi ketinggian maka tekanan udara akan semakin rendah. Semakin rendah ketinggian maka tekanan udara akan semakin tinggi.



Gambar 2.20 Sensor BMP280

Pada umumnya pengukuran ketinggian dengan informasi tekanan udara menggunakan standar tekanan udara di permukaan laut, sebesar 101.325 kPa (760 mmHg). Setiap kenaikan ketinggian 10 meter akan menurunkan tekanan udara sebesar 1 mmHg. Berikut formula umumnya:

$$h = (Po - P)\ x\ 10\ (meter) \tag{2.7}$$

Po adalah tekanan udara standar permukaan laut (mmHg), P adalah tekanan udara terukur posisi suatu benda (mmHg) dan h adalah ketinggian dari posisi benda (tiap 10 meter).

2.9 *Global Positioning System* (GPS) Ublox 6M

GPS adalah adalah sebuah sistem navigasi berbasiskan radio yang menyediakan informasi koordinat posisi, kecepatan, dan waktu, ketinggian dan informasi tambahan lainnya kepada pengguna di seluruh dunia berdasarkan perbedaan garis lintang (*latitude*) dan garis bujur (*longitude*). Pengguna hanya membutuhkan GPS *receiver* untuk dapat mengetahui koordinat lokasi. Keakuratan koordinat lokasi tergantung pada tipe GPS *receiver*. GPS terdiri dari tiga bagian yaitu satelit pemancar yang mengorbit bumi (Satelit GPS mengelilingi bumi 2 kali

sehari), stasiun pengendali dan pemantau di bumi, dan GPS *receiver*. Alat penerima GPS inilah yang dipakai oleh pengguna untuk melihat koordinat posisi.

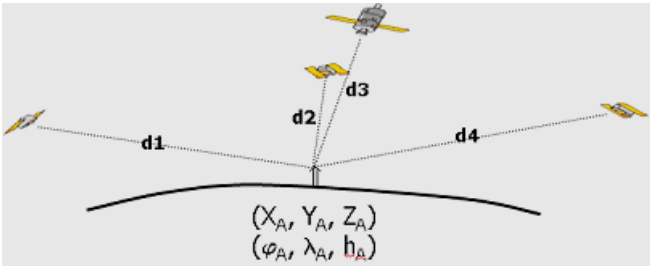
Satelit GPS memancarkan dua sinyal yaitu frekuensi L1 (1575.42 MHz) dan L2 (1227.60 MHz). Sinyal L1 dimodulasikan dengan dua sinyal pseudo-random yaitu kode P (*Protected*) dan kode C/A (*coarse/aquisition*). Sinyal L2 hanya membawa kode P. Setiap satelit mentransmisikan kode yang unik sehingga penerima dapat mengidentifikasi sinyal dari setiap satelit. Pada saat fitur "*Anti-Spoofing*" diaktifkan, maka kode P akan dienkripsi dan selanjutnya dikenal sebagai kode P(Y) atau kode Y.

Tipe GPS yang digunakan pada Tugas Akhir ini adalah GPS Ublox 6M yang memiliki spesifikasi sebagai berikut :

Tabel 2.2 Spesifikasi GPS Ublox 6M

Parameter	Spesifikasi
Receiver type	50 *Channels* GPS L1 *frequency*, C/A Code SBAS: WAAS, EGNOS, MSAS
Time-To-First-Fix	*Cold start* : 27 s *Warm start* : 27 s *Hot Start* : 1 s
Sensitivity	*Tracking & Navigation* :161 *Reacquisition* : 160 dB *Cold Start (without aiding)* : 146 dBm *Hot Start* : 156 dBm
Maximum Navigation update rate	5Hz
Horizontal position accuracy	GPS : 2.5 m SBAS : 2.0 m
Heading accuracy	0.5 *degrees*
Operational Limits	*Dynamics* : 4 g *Altitude* : 50,000 m *Velocity* : 10 500 m/s

Prinsip dasar penentuan posisi dengan GPS adalah perpotongan ke belakang dengan pengukuran jarak secara simultan ke beberapa satelit GPS seperti gambar berikut :



Gambar 2.21 Prinsip dasar GPS

Kumpulan satelit-satelit berada di orbit bumi, sekitar 12.000 mil diatas permukaan bumi. Kumpulan satelit ini diatur sedemikian rupa sehingga alat navigasi setiap saat dapat menerima paling sedikit sinyal dari empat buah satelit. Sinyal satelit ini bersifat *line of sight* (sebatas mata memandang) yang artinya dapat melewati awan, kaca, atau plastik, tetapi tidak dapat melewati gedung atau gunung. Satelit mempunyai jam atom, dan juga akan memancarkan informasi waktu. Data ini dipancarkan dengan kode '*pseudo-random*'. Masing-masing satelit memiliki kodenya sendiri-sendiri. Nomor kode ini biasanya akan ditampilkan di alat navigasi, maka kita bisa melakukan identifikasi sinyal satelit yang sedang diterima alat tersebut. Data ini berguna bagi alat navigasi untuk mengukur jarak antara alat navigasi dengan satelit, yang akan digunakan untuk mengukur koordinat lokasi. Kekuatan sinyal satelit juga akan membantu alat dalam penghitungan. Kekuatan sinyal ini lebih dipengaruhi oleh lokasi satelit, sebuah alat akan menerima sinyal lebih kuat dari satelit yang berada tepat diatasnya.

Satelit akan memancarkan data almanak dan *ephemeris* yang akan diterima oleh alat navigasi secara teratur. Data almanak berisikan perkiraan lokasi (*approximate location*) satelit yang dipancarkan terus menerus oleh satelit. Data *ephemeris* dipancarkan oleh satelit, dan valid untuk sekitar 4-6 jam. Untuk menunjukkan koordinat sebuah titik (dua

dimensi), alat navigasi memerlukan paling sedikit sinyal dari 3 buah satelit. Untuk menunjukkan data ketinggian sebuah titik (tiga dimensi), diperlukan tambahan sinyal dari 1 buah satelit lagi. Dari sinyal-sinyal yang dipancarkan oleh kumpulan satelit tersebut, alat navigasi akan melakukan perhitungan-perhitungan, dan hasil akhirnya adalah koordinat posisi alat tersebut. Makin banyak jumlah sinyal satelit yang diterima oleh sebuah alat, akan membuat alat tersebut menghitung koordinat posisinya dengan lebih tepat.

Tingkat akurasi GPS ditentukan dengan istilah GDOP (*Geometric Dilution Of Precision*). DOP(*Dilution Of Precision*) atau GDOP adalah istilah yang digunakan dalam navigasi satelit dan geomatika untuk menentukan efek tambahan geometri satelit navigasi pada tingkat kepresisian pengukuran posisi. Ketika satelit GPS yang tertangkap oleh *receiver* dalam posisi berdekatan di ruang angkasa, maka nilai geometrinya lemah dan nilai DOP tinggi. Saat dalam posisi berjauhan maka nilai geometrinya kuat dan nilai DOP nya rendah. Dengan demikian nilai DOP rendah merupakan tingkat kepresisian posisi yang lebih baik karena pemisahan sudut yang lebih luas antara satelit yang digunakan untuk menghitung posisi pengguna. Faktor-faktor lain yang dapat meningkatkan DOP efektif adalah penghalang seperti pegunungan atau bangunan di dekatnya. DOP dapat dinyatakan sebagai jumlah pengukuran yang terpisah. Istilah lebih spesifik mengenai DOP yaitu HDOP (*Horizontal Dilution Of Precision*), VDOP (*Vertical Dilution Of Precision*) dan TDOP (*Time Dilution Of Precision*). Pengaruh geometri satelit pada kesalahan posisi disebut *dilution of precision* dan itu diartikan sebagai rasio *position error* terhadap *range error*. Dapat dibayangkan jika piramida persegi dibentuk oleh garis bergabung empat satelit dengan penerima di ujung piramida. Semakin besar volume piramida maka semakin baik (lebih rendah) nilai GDOP sedangkan jika volumenya lebih kecil maka lebih buruk (lebih tinggi) nilai GDOP nya. Demikian pula semakin banyak jumlah satelit yang ditangkap *receiver* maka semakin baik nilai GDOP nya. Tabel nilai DOP yang mempengaruhi tingkat kepresisian GPS adapat dilihat pada tabel berikut.

Tabel 2.3 Tingkat kepresisian GPS berdasarkan *datasheet*

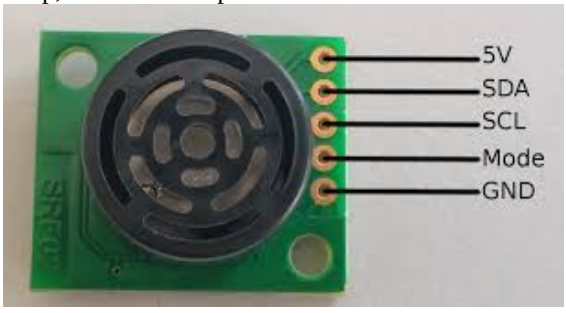
Nilai DOP	Nilai kepresisian	Keterangan
< 1	*Ideal*	Level tertinggi tingkat kepresisian GPS yang digunakan untuk keperluan sistem performa keakurasian tinggi pada setiap saat.
1 -2	*Excellent*	Level tingkat keakurasian tinggi untuk sistem aplikasi yang sensitif.
2 -5	*Good*	Level kepresisian yang sedang. Sering digunakan untuk keperluan navigasi pada umumnya.
5 -10	*moderate*	Level kepresisian kurang baik tetapi masih dapat digunakan sebagai perhitungan.
10 – 20	*Fair*	Level kepresisian yang rendah, hanya untuk perkiraan posisi.
> 20	*Poor*	Level kepresisian sangat rendah. Ketidakakuratan posisi bisa mencapai 300 m.

2.10 Ultrasonoc SRF02

SRF02 merupakan sensor ultrasonic yang mempunyai pancaran frekuensi 40 Khz, Pemancar dan penerima SRF02 ini sudah tergabung menjadi satu. Koneksi SRF02 terhadap mikrokontroler pada pembuatan proyek akhir ini dilakukan dengan mode I2C.

Agar bekerja pada mode I2C maka pin mode tidak diberi logika atau dihubungkan ke ground. Modul SRF02 hanya akan melakukan proses pengukuran jarak jika telah diberikan perintah terlebih dahulu. Dengan mengirimkan data serial yang berisi alamat sensor kemudian diikuti dengan data 0 (0H) sebagai perintah maka SRF02 akan mulai melakukan proses pengukuran jarak dan setelah selesai dengan segera

akan mengirimkan data 2 byte sebagai data hasil pengukurannya. Data 2 byte hasil pengukuran ini akan dikirimkan byte atasnya (upper byte) terlebih dahulu kemudian diikuti byte rendahnya (lower byte). Jika data perintahnya 81 (0x51) maka hasil pengukurannya akan memiliki satuan centimeter. Sesuai dengan spesifikasi yang diberikan oleh produsen modul SRF02 ini, mode I2C ini hanya dapat bekerja pada kecepatan transmisi (baud rate) 9600 bps dan dengan format data 1-bit start, 8-bit data, 2-bit stop, dan tidak ada paritas.



Gambar 2.22 SRF02

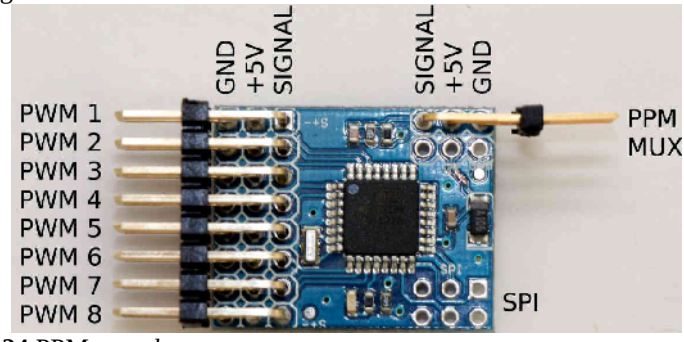
2.11 MPXV7002DP

MPXV7002 adalah sensor takanan silikon monolitik yang dirancang untuk berbagaimacam aplikasi terutama mikrokontroler atau mikroprosesor dengan input analog digital dan pengolahan membandingkan dua tekanan untuk keakuratan sinyal serta diperuntukan untuk membaca tekanan positif dan negatif. Dengan offset 2,5V sampai 0V untuk mengukur tekanan hingga 7000 Pa dan bisa untuk menghitung kecepatan di udara dengan catu daya 5,25 volt sampai 4,75 volt.

Gambar 2.23 MPXV7002DP *airspeed*

2.12 PPM Encoder

PPM *encoder* ini akan mengkodekan input PWM biasa menjadi output PPM tunggal. Hal ini memungkinkan untuk menggabungkan hingga 8 chanel dari penerima remot kontrol untuk input PPM tunggal untuk controller penerbangan atau proyek elektronik, dengan menggunakan processor ATMEL ATmega328P AVR. Dengan begitu kita tidak perlu membutuhkan banyak pin input pada mikrokontroler arduino mega.



Gambar 2.24 PPM *encoder*

[Halaman ini sengaja dikosongkan]

BAB 3
PERANCANGAN DAN REALISASI ALAT

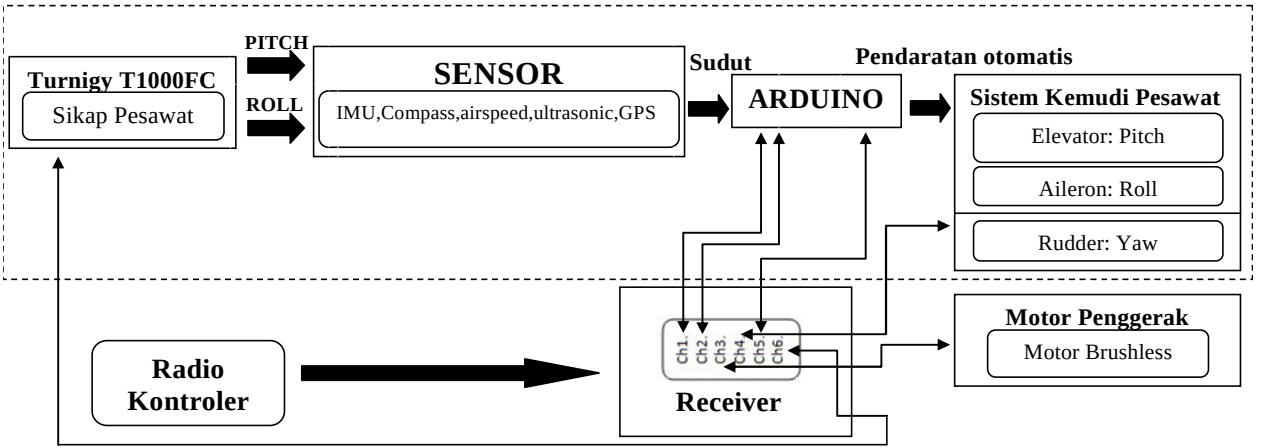
Pada bab ini menjelaskan tentang perancangan dan implementasi sistem meliputi arsitektur sistem, perancangan *hardware*, perancangan *software*, desain kontrol PID dan persiapan pesawat.

3.1 Arsitektur Sistem

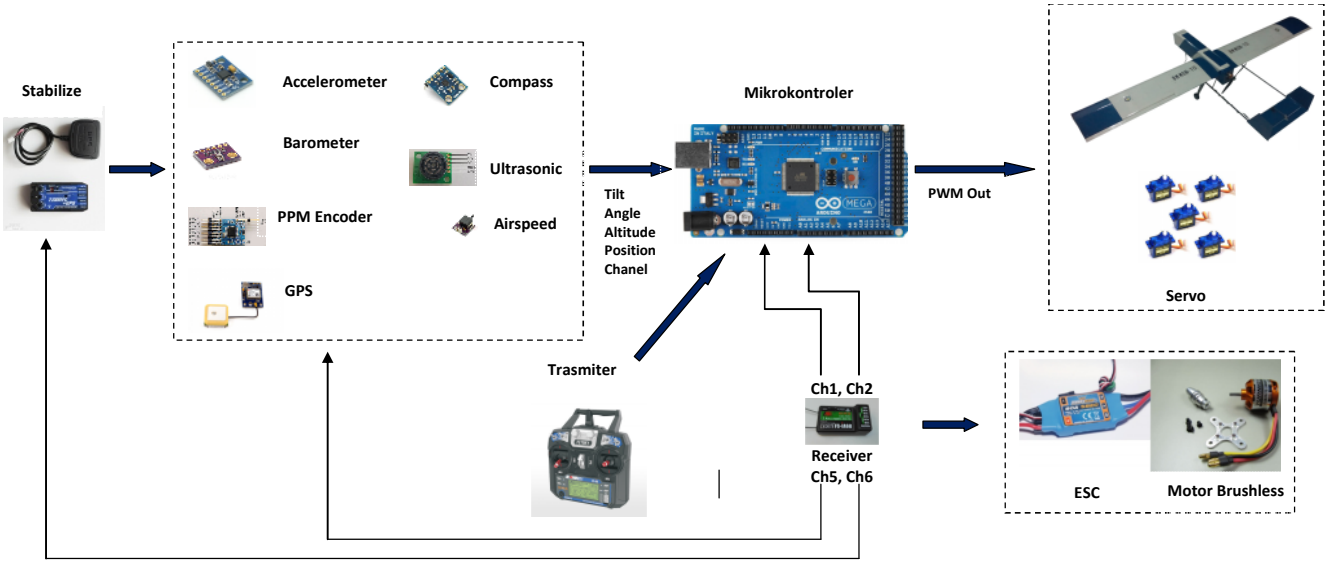
Blok diagram diatas merupakan penyerderhaan dari keseluruhan sistem pendaratan otomatis pada UAV dengan menggunakan GPS *waypoint* yang dirancang pada pelaksanaan tugas akhir ini. Perancangan sistem ini bertujuan untuk mengendalikan UAV baik secara manual maupun otomatis mengikuti *waypoint* yang diinginkan dan mendarat otomatis. Mikrokontroler digunakan sebagai kontroler kendali pada pesawat dan pengolah data dari seluruh sensor dan *receiver*. Sistem yang akan dirancang berfungsi untuk mengendalikan pergerakan servo agar pesawat memiliki mampu terbang dalam kondisi stabil dan dapat bergerak mengikuti waypoint GPS yang diinginkan.

Pada kontrol manual, mikrokontroler akan membaca sinyal PWM dari *receiver* kemudian diolah dan selanjutnya dikirim menuju servo aileron dan elevator sedangkan signal PWM rudder dan motor DC *brushless* yang ditangkap oleh receiver langsung menuju servo tanpa masuk melalui mikrokontroler terlebih dahulu. Pada saat kontrol otomatis dijalankan, mikrokontroler akan memproses data-data dari sensor *gyroscope*, *accelerometer* pada modul turnigy T1000FC sehingga pesawat akan memperoleh sikap kestabilan di udara. Selanjutnya mikrokontroler memroses data dari sensor *barometric altimeter*, *ultrasonic* sehingga pesawat dapat menjaga setpoint ketinggian yang diinginkan untuk melaksanakan pendaratan otomatis. Sebagai alat navigasi di udara, pesawat menggunakan GPS dan kompas yang mana pesawat akan mengikuti waypoint garis lintang dan garis bujur yang dimasukkan ke dalam program pada saat pesawat sebelum terbang. Sensor *gyroscope* dan *accelerometer* pada turnigy T1000FC akan menghasilkan data sudut kemiringan pesawat, sedangkan sensor *barometric altimeter*, *ultasonic* mengahsilkan data ketinggian, GPS dan kompas menghasilkan data berupa posisi pesawat berdasarkan garis

lintang dan garis bujur serta dapat memberikan data *heading* pesawat pada saat pesawat mengalami perpindahan posisi. Hasil dari pengolahan data tersebut diaktualisasikan dalam pergerakan servo (aileron dan elevator). Kecepatan motor DC *brushless* dan rudder diatur secara manual.



Gambar 3.1 Diagram blok arsitektur sistem



Gambar 3.2 Perancangan *hardware* sistem

3.2 Perancangan *Hardware*

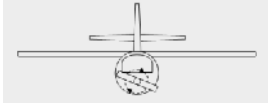
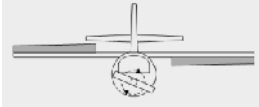
Perancangan *hardware* sistem kontrol UAV berupa penggabungan fungsi beberapa komponen yaitu pesawat terbang, radio kontrol, mikrokontroler, sensor dan aktuator.

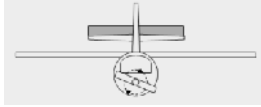
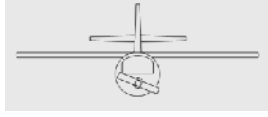
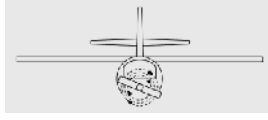
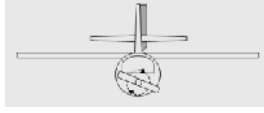
3.2.1 Sinkronisasi Radio Kontrol dan Pergerakan Aktuator

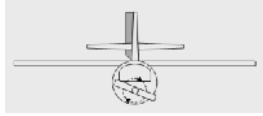
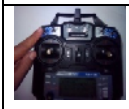
Radio kontrol pada tugas akhir ini berfungsi sebagai kontrol pergerakan aktuator pesawat dan saklar manual-otomatis. Hal pertama yang dilakukan yaitu pengecekan gerak aktuator setiap terjadi pergeseran kemudi pada radio kontrol secara manual. Kemudian perubahan lebar pulsa sinyal kontrol nilai minimum, nilai tengah dan nilai akhir dicatat sebagai dasar acuan pergerakan servo sebagai aktuator dalam kondisi otomatis.

Berikut data perubahan sinyal radio kontrol dan lebar pulsa terhadap pergerakan aktuator pesawat:

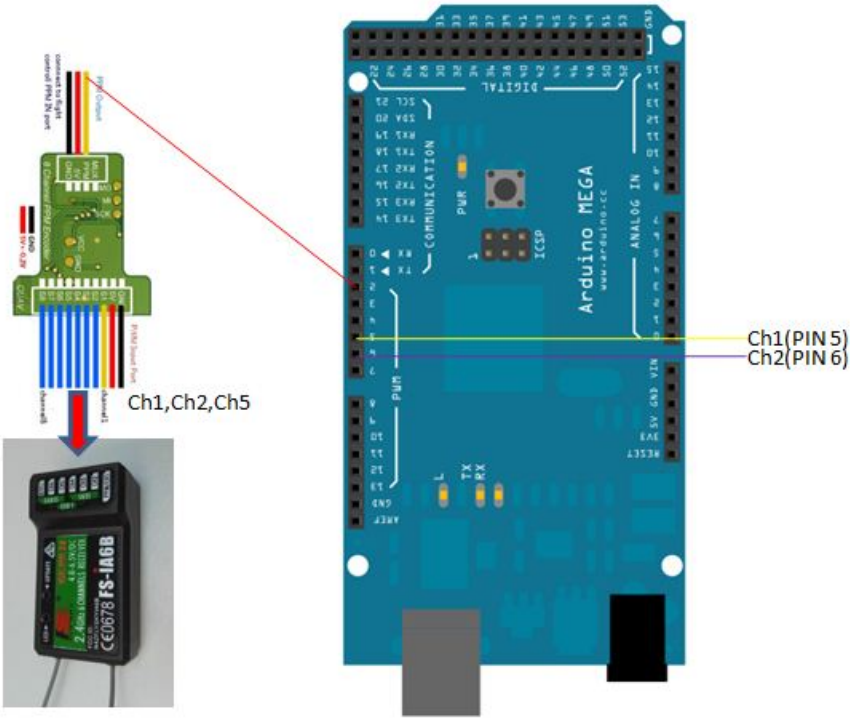
Tabel 3.1 Konfigurasi Radio Kontrol dan Aktuator

Radio Kontrol	Lebar Sinyal Pulsa (µs)	Aktuator Pesawat
	Ch-1 = 1500 Ch-2 = 1500 Ch-3 = 1500 Ch-4 = 1500 Ch-5 = 1000	
	Ch-1 = 1000 Ch-2 = 1500 Ch-3 = 1500 Ch-4 = 1500 Ch-5 = 1000	
	Ch-1 = 2000 Ch-2 = 1500 Ch-3 = 1500	

	Ch-4 = 1500 Ch-5 = 1000	
	Ch-1 = 1500 Ch-2 = 1000 Ch-3 = 1500 Ch-4 = 1500 Ch-5 = 1000	
	Ch-1 = 1500 Ch-2 = 2000 Ch-3 = 1500 Ch-4 = 1500 Ch-5 = 1000	
	Ch-1 = 1500 Ch-2 = 1500 Ch-3 = 1000 Ch-4 = 1500 Ch-5 = 1000	
	Ch-1 = 1500 Ch-2 = 1500 Ch-3 = 2000 Ch-4 = 1500 Ch-5 = 1000	
	Ch-1 = 1500 Ch-2 = 1500 Ch-3 = 1500 Ch-4 = 1000	

	Ch-5 = 1000	
	Ch-1 = 1500 Ch-2 = 1500 Ch-3 = 1500 Ch-4 = 2000 Ch-5 = 1000	
	Ch-1 = 1500 Ch-2 = 1500 Ch-3 = 1500 Ch-4 = 1500 Ch-5 = 1000	Kontrol Manual - ON
	Ch-1 = 1500 Ch-2 = 1500 Ch-3 = 1500 Ch-4 = 1500 Ch-5 = 2000	Kontrol Otomatis - ON

Tabel 3.1 menjelaskan tentang perubahan sinyal radio kontrol dan lebar pulsa terhadap pergerakan kemudi pesawat baik aileron, elevator dan rudder. Pada mode otomatis, pesawat hanya menggunakan kemudi *aileron* untuk pergerakan rolling dan elevator untuk pergerakan pitching ketika mengudara. Aktuasi pergerakan *throttle* dan *rudder* dijalankan secara manual, yaitu signal PWM dari receiver langsung *bypass* menuju servo aktuator tanpa melalui mikrokontroller terlebih dahulu.



Gambar 3.3 Koneksi rangkaian PPM encoder dan Arduino Mega2560

Tabel 3.2 Koneksi Pin Sensor PPM encoder dan Arduino Mega2560

PPM encoder		Arduino Mega 2560	
Pin	Fungsi	Pin	Fungsi
1	Vcc	-	-
2	GND	-	-
3	Input	2	PPM encoder
4	Ch1	5	Aileron

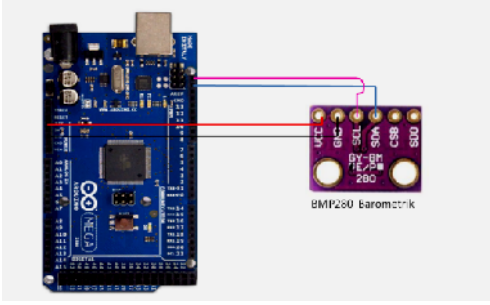
PPM encoder		Arduino Mega 2560	
Pin	Fungsi	Pin	Fungsi
5	Ch2	6	Elevator

3.2.2 Rangkaian Sensor IMU

Sensor IMU menggunakan turnigy T1000FC yang didalamnya terdapat *accelerometer* dan *gyro* yang berfungsi sebagai pengatur sikap pesawat agar dalam keadaan level, output dari IMU tersebut masuk ke dalam mikrokontroler untuk diolah pergerakan *aileron* dan *elevator*.

3.2.3 Perancangan Rangkaian Sensor Ketinggian

Sensor ketinggian menggunakan sensor barometrik BMP280. Rangkaian sensor IMU terkoneksi dengan mikrokontroler melalui komunikasi i2c (gambar 3.4).



Gambar 3.4 Koneksi rangkaian BMP280 dan mikrokontroler

Tabel 3.3 Konfigurasi Pin Sensor BMP280- mikrokontroler

BMP280		Arduino Mega2560	
Pin	Fungsi	Pin	Fungsi
1	Vcc	-	-

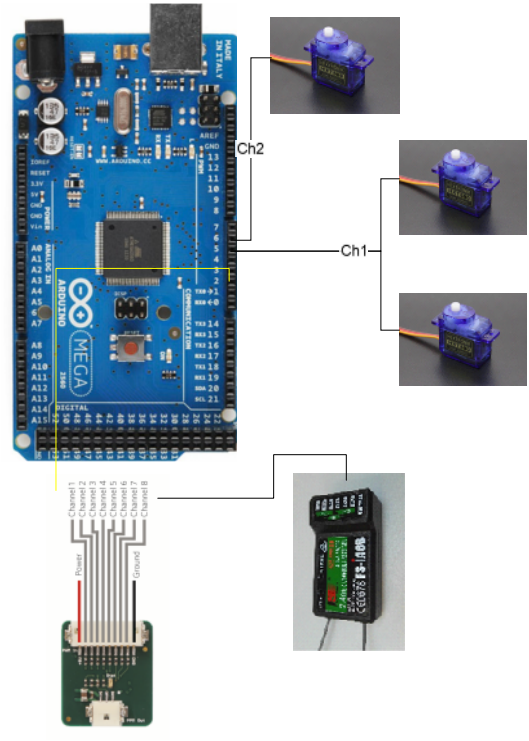
BMP280		Arduino Mega2560	
Pin	Fungsi	Pin	Fungsi
2	GND	-	-
3	SCL	SCL	SCL I2C
4	SDA	SDA	SDA I2C

3.2.4 Perancangan Rangkaian Input dan output PWM

Arduino mega 2560 berfungsi untuk menangkap sinyal input PWM dari *receiver* kemudian diolah oleh mikrokontroler untuk diteruskan menuju servo.

Tabel 3.4 Konfigurasi Pin *Receiver* – mikrokontroler

Receiver		Arduino Mega 2560	
Pin	Fungsi	Pin	Fungsi
Ch-1	Aileron	PWM 5	PWM input-1
Ch-2	Elevator	PWM 6	PWM input-2
Ch-3	Throttle	*Bypass*	-
Ch-4	Rudder	*Bypass*	-
Ch-5	Saklar	PPM encoder	PWM input-5
Ch-6	T1000FC	*Bypass*	*Autopilot*



Gambar 3.5 Koneksi input-output PWM

3.3 Perancangan *Software*

Perancangan *software* dibutuhkan untuk mengkoneksikan komponen-komponen elektronik agar sistem dapat berjalan dengan baik. Perancangan meliputi pengolahan output dari stabilizer turnigy T1000FC, penunjukan arah (*heading*) dari GPS dan kompas, penentuan garis lintang dan garis bujur dari GPS, sudut ke *waypoint* yang dituju dan ketinggian dari *baseline* dari sensor *barometric altimeter, ultrasonic* dan pengolahan PWM yang selanjutnya diteruskan ke servo aktuator.

3.3.1 Penentuan Waypoint

Data dari sensor *accelerometer* dan *gyroscope* output dari turnigy T1000FC yang telah diolah menjadi kesetabilan pesawat, maka output akan diolah menjadi pergerakan pesawat menuju titik titik yang ditentukan berdasarkan koordinat yang telah dimasukkan ke dalam program.

3.3.2 PWM Input dan Output Kontroler

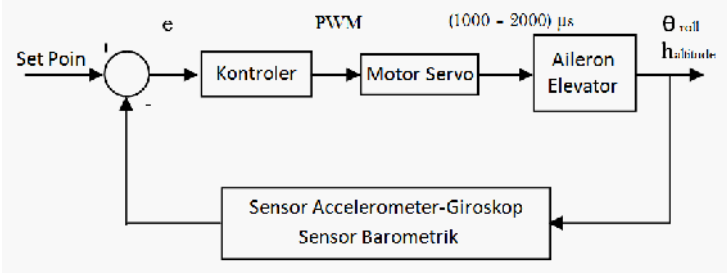
Pembacaan signal PWM dari receiver tersebut dengan memasukkan input PWM channel 1, channel 2 dan channel 5 ke pin PPM encoder 1,2 dan 5 serta dikeluarkan ke pin 2. Setelah masuk ke pin 2 maka akan dikeluarkan ke pin 5 dan 6 sebagai penggerak *aileron* dan *elevator*. Program pembacaan input PPM terdapat pada *coding program* pada lampiran B

Aktualisasi proses kontrol adalah untuk mengatur sudut gerak servo baik pada servo aileron, elevator maupun rudder

Nilai sudut yang diinputkan pada servo dapat diatur sesuai dengan apa yang kita inginkan dengan tetap mengacu pada spesifikasi teknik dari servo itu sendiri.

3.4 Desain Kontroler Proporsional-Integral pada Kestabilan Pesawat

Algoritma kontrol proporsional integral pada sistem kestabilan pesawat meliputi sudut *pitch* dan *roll*. Desain kontroler PI digunakan ketika mode otomatis diaktifkan. Pada kontrol kestabilan pesawat di udara, *set point* kontroler PI menggunakan referensi sudut nol *pitch* dan *roll*. Apabila terjadi perubahan kemiringan pada saat pesawat mengudara, akan menghasilkan error *pitch* dan *roll*. *Error* tersebut kemudian diproses pada kontroler, sehingga menghasilkan sinyal PWM untuk menggerakkan aileron atau elevator. Pergerakan aileron atau elevator akan memberikan kompensasi terhadap kemiringan pesawat, sehingga diharapkan sudut *pitch* dan *roll* bernilai nol (pesawat terbang lurus dan stabil).



Gambar 3.6 Kontrol PI kestabilan pesawat *pitch* dan *roll*

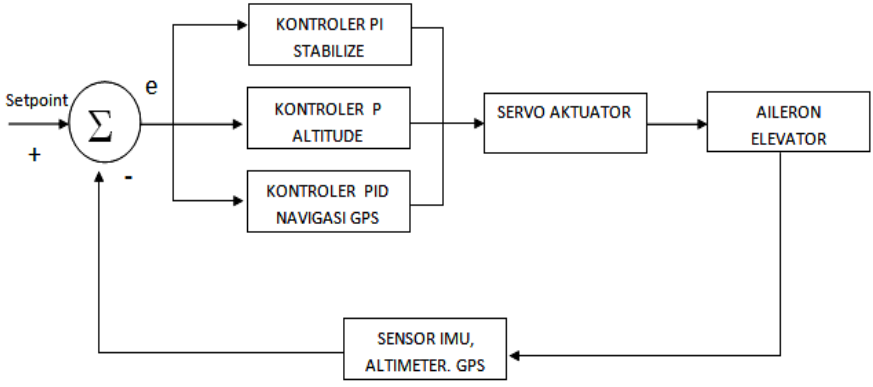
3.5 Desain Kontroler Proporsional pada *altitude*

Sistem kestabilan pesawat di udara dengan menggunakan *gyro* dan *accelerometer* belum cukup untuk digunakan sebagai *flight controller* pada pesawat terbang. Pesawat terbang harus mampu mempertahankan ketinggian pada saat di udara karena pergerakan plant pada saat sistem kontrol belum mencapai keadaan *steady state* mengakibatkan pesawat mengalami penurunan atau kenaikan ketinggian. Disamping itu bentuk aerodinamika pesawat sangat mempengaruhi sistem dalam mencapai keadaan stabil. Untuk mempertahankan ketinggian pesawat maka digunakan sensor *barometric pressure* sebagai sensor ketinggian barometrik. *Setpoint* ketinggian ditentukan pada saat perubahan switch channel 5. Pada saat *switch* tersebut memberikan input PWM diatas 1500us, maka *mode* otomatis akan dijalankan, begitu pula sebaliknya. *Listing* program pada kontrol ketinggian berada pada Lampiran B

3.6 Desain Kontroler Proporsional-Derivatif pada Navigasi menggunakan GPS dan kompas *Waypoint*

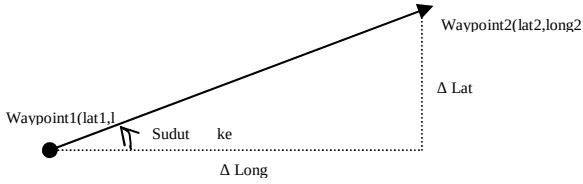
Navigasi adalah suatu cara untuk mengarahkan pesawat dari suatu titik ke titik lainnya dengan arah tertentu. Untuk dapat menggunakan sistem navigasi GPS, sistem harus dapat mendeteksi valid atau tidaknya GPS *receiver* dalam menerima signal GPS dari beberapa satelit. Disamping itu nilai HDOP sangat menentukan tingkat keakurasian GPS dalam menentukan posisi seperti yang telah diuraikan pada bab sebelumnya. Dalam perancangan Tugas Akhir ini jumlah satelit minimal yang dibutuhkan untuk menjalankan mode navigasi otomatis

adalah sebanyak 4 satelit. Untuk menentukan arah(*heading*) pesawat terhadap 0° (arah utara) maka diguanakanlah *tilt compesation* yaitu gabungan antara kompas dan *accelerometer*. *Heading* pesawat didapat pada saat pesawat mengalami perpindahan posisi (*latitude* dan *longitude*). Sudut dan jarak estimasi dari titik awal ke titik tujuan didapat dari perbedaan posisi awal dan titik *latitude-longitude* tujuan yang aka dituju. Dengan demikian maka didapatkan *setpoint* arah ke waypoint yang diinginkan. Perbedaan besaran derajat antara sudut ke tujuan dan arah heading saat ini didefinisan sebagai *error* arah *waypoint.* Selanjutnya nilai *error* tersebut digunakan untuk pengaturan sistem agar pesawat dapat membelok sesuai sudut pesawat ke tujuan. Pada sistem pesawat, nilai *output* dari kontrol proporsinal-derivatif diimplementasikan pada pengaturan sudut aileron. *Listing* program penentuan jarak ke tujuan adalah sebagai berikut :



Gambar 3.7 Kontrol keseluruhan sistem navigasi otomatis

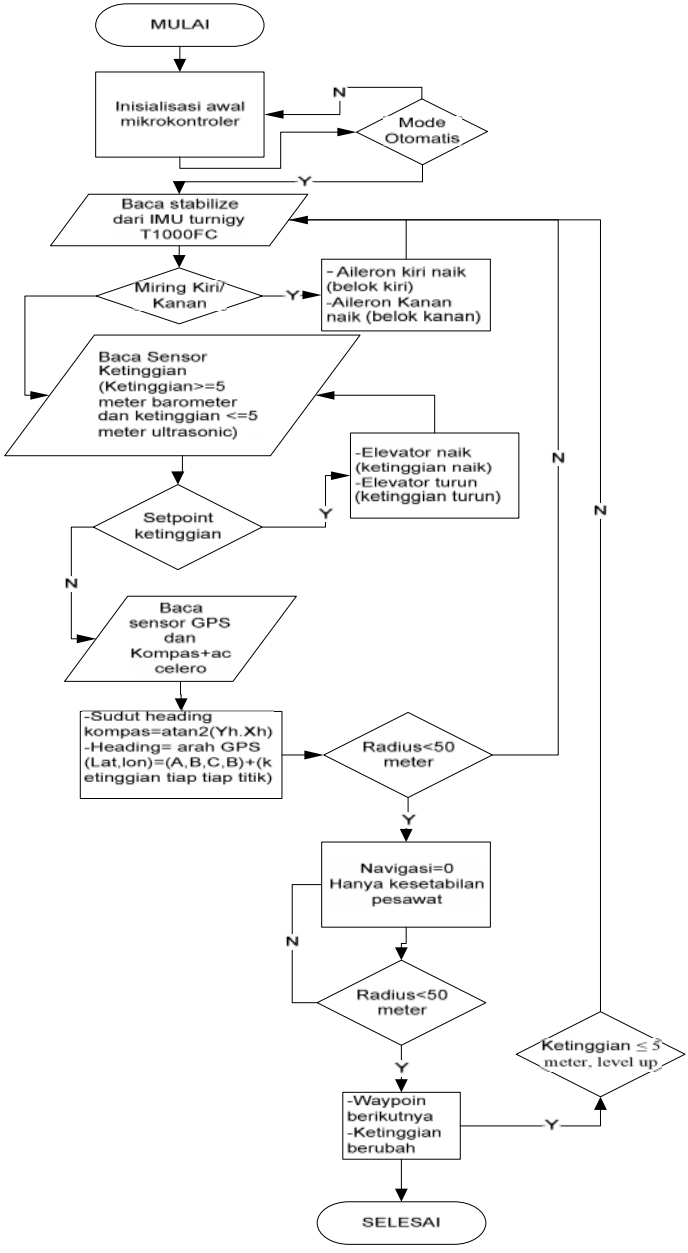
Sedangkan untuk menentukan sudut ke arah tujuan didapatkan dari selisih *latitude* dan *longitude* antara posisi tujuan dan posisi awal. Seccara matematis nilai sudut didapatkan dengan rumus *arctan* dari selisih *latitude* dibagi dengan selisih *longitude* nya. Sedangkan jarak ke tujuan diperoleh dengan mencari nilai kemiringan baik dengan *phytagoras* ataupun dengan perhitungan *sinus-cosinus* dengan sketsa perhitungan seperti pada gambar 3.8.



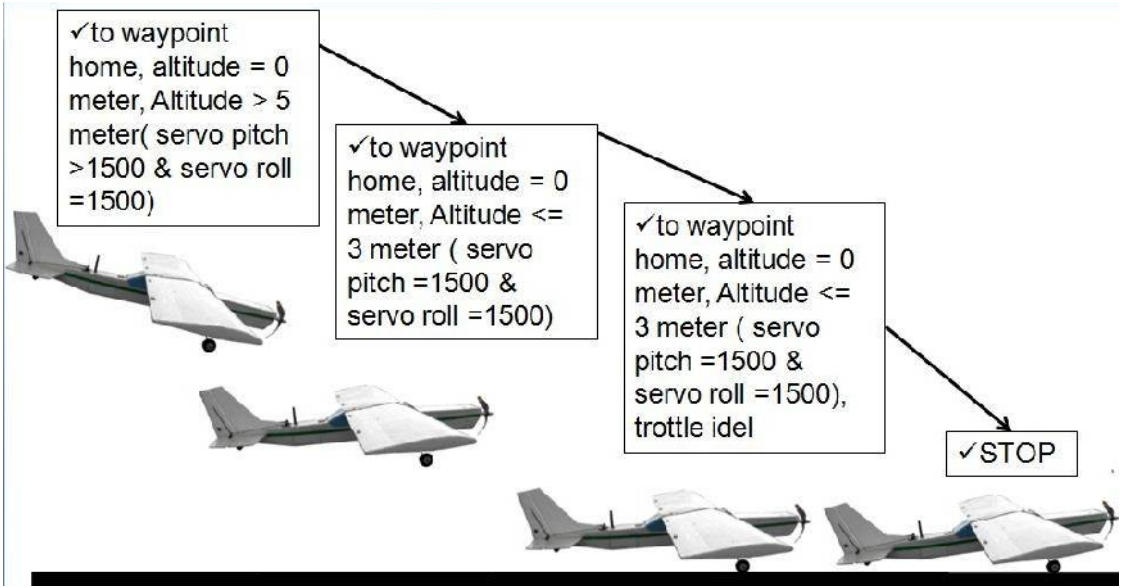
Gambar 3.8 Sketsa perhitungan jarak dan sudut ke tujuan

Listing program penentuan jarak ke tujuan dan arah sudut ke tujuan sudah ada pada *library tilt compesation* yang bersifat *open source*. Listing programnya pada lampiran B :

Tahap selanjutnya adalah pembuatan program untuk sistem kontrol navigasi pesawat menggunakan GPS *waypoint*. Untuk mengarahkan pesawat agar mempunyai lokasi tujuan maka *latitude* dan *longitude* tujuan harus di deklarasikan pada awal program. Agar seluruh listing program mantik dan dapat berjalan secara efektif dan efisien maka diperlukan algoritma/ *flowchart* bahasa pemrogaman seperti pada gambar 3.9.



Gambar 3.9 *Flowchart* pemrogaman



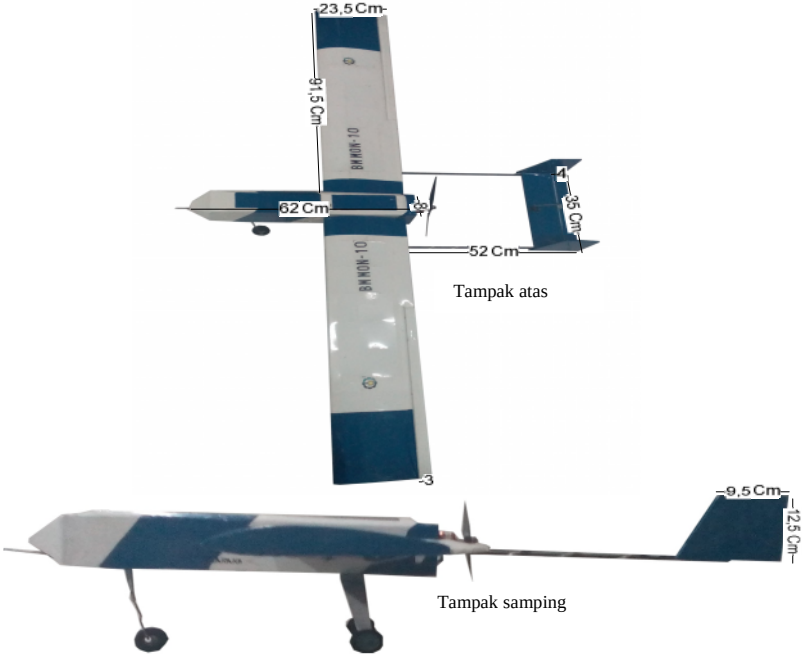
Gambar 3.10 *Algoritma* saat pendaratan

3.7 Persiapan Pesawat

Persiapan pesawat meliputi desain dan pembuatan pesawat UAV *fix wing* dan penempatan sistem kontrol.

3.7.1 Pembuatan Pesawat UAV

Pesawat UAV menggunakan desain dengan model yang dibuat sendiri oleh penulis. Model pesawat yang digunakan dalam perancangan sistem ini adalah *high wing trainer*. Pesawat tersebut memungkinkan pergerakan pesawat dapat dikontrol dan stabil ketika mengudara. Pesawat menggunakan 4 buah servo yang berfungsi untuk menggerakkan aileron, elevator dan rudder. Sebuah motor DC *brushless* berfungsi sebagai penghasil daya dorong pesawat. Motor DC *brushless* terhubung dengan ESC sebagai pengontrol kecepatan motor untuk memutar baling-baling.



Gambar 3.11 UAV *Midle Wing*

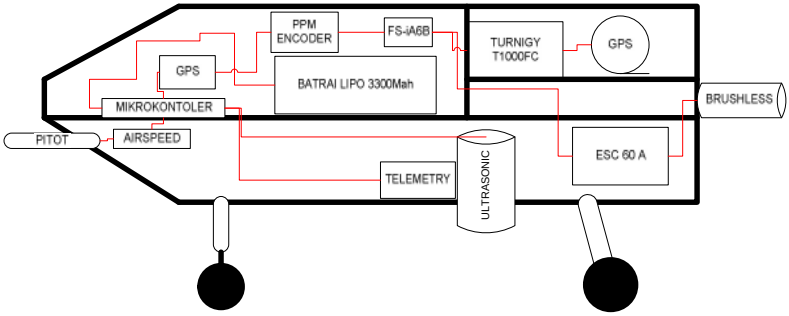
Tabel 3.5 Spesifikasi UAV *high wing trainer*

Fitur	Keterangan
Tipe	Pesawat latih
Bentang sayap	18300 mm
Panjang pesawat	11400 mm
Wing Chord	2350 mm
Bobot total	1875 gram
Material	Kayu balsa
ESC	60 A
Tenaga penggerak	*Single DC brushless motor*
Tipe motor *brushless*	Turnigy 3530 /1400 KV
Catudaya	11.1V 3300mAh 3S 20C Li-Polymer
Channel	buah (*aileron, elevator,throttle,rudder*)
Motor Servo	gram, V = 0.15 sec/60°, torsi 3.7 kg.cm

3.7.2 Penempatan Kontroler pada Pesawat

Pesawat memiliki *center of gravity* (CG), sebagai keseimbangan pesawat saat mengudara. Spesifikasi CG pesawat *high wing* yaitu terletak pada seperempat depan *leading edge* sayap pesawat. Penempatan kontroler harus menjaga beban kesetimbangan pesawat tetap pada titik CG.

Dimensi ruang pada badan pesawat yang kecil memiliki keterbatasan dalam memuat komponen. Oleh karena itu dibutuhkan penempatan kontroler tanpa mengganggu sistem kerja pesawat.



Gambar 3.12 Penempatan kontroler pada pesawat

BAB 4
PENGUJIAN DAN ANALISA SISTEM

Bab ini menjelaskan tentang hasil pengujian dan analisa dari perancangan sistem yang telah dilaksanakan dan dijelaskan pada bab sebelumnya. Pengujian dari sistem kontrol ini dilakukan menjadi dua tahap, yaitu pengujian parsial dan pengujian integrasi. Pengujian parsial berupa pengujian ketinggian dari *barometer* dan *ultrasonic* pesawat, sinyal kontrol PWM, kompas dan kontrol navigasi GPS. Pengujian terintegrasi meliputi pengujian sistem navigasi otomatis dengan *waypoint* GPS dan ketinggian yang dimasukkan.

4.1 Pengujian Parsial

Pengujian dilakukan pada sub-sistem kontrol yang telah terpasang pada UAV.

4.1.1 Pengujian ketinggian

Pengujian ini dimaksudkan untuk mengetahui respon ketinggian di udara pada saat sensor *ultrasonic* membaca ketinggian 5 meter maka ketinggian diambil alih oleh sensor *ultrasonik* sedangkan selebihnya akan diambil alih oleh sensor *barometer*.

Tabel 4.1 Pembacaan *Barometer*

Gambar 4.1 Pengukuran *error barometer*

Tabel 4.2 Pembacaan *Ultrasonic*

Gambar 4.2 *Error ultrasonic* mesin mati dan menyala

4.1.2 Pengujian Sinyal PWM pada Mikrokontroler

Pengujian sinyal PWM dilaksanakan dengan membaca nilai PWM dalam spesifikasi *time high* (t_{on}) pada setiap *channel* radio kontrol. Pengujian dilaksanakan pada saat nilai PWM minimum, nilai tengah dan nilai maksimum. Nilai tengah rata-rata dari *channel* 1,2,3,4 adalah 1500 µs. Nilai PWM minimum rata-rata adalah 1000us dan nilai PWM maksimumnya adalah 2000 us. Nilai PWM pada saat *idle* (*control stick*) tidak bergerak mempunyai sedikit pergeseran kurang lebih berkisar antara 1 sampai dengan 30 us. Hal ini mengakibatkan getaran kecil (*glitch*) pada servo aktuator di waktu yang lama.

Untuk merubah nilai PWM menjadi pergerakan sudut servo dalam derajat, maka nilai PWM yang terbaca mikrokontroler maka nilai PWM 1500 ke 2000 adalah 45^o bila akan mengurangi sudut servo maka tinggal dikurangi nilai PWMnya:

Tabel 4.3 Pengujian PWM

Gambar 4.3 Grafik pengukuran PWM

4.1.3 Pengujian Kompas

Dalam pembacaan kompas arah utara dinyatakan dengan nilai 0^o atau 360^o dan arah selatan 180^o secara parsial kita mencoba untuk pengujian kompas yang digabung dengan accelerometer agar pesawat tetap bisa membaca kompas meskipun dalam keadaan belok

Tabel 4.4 Pembacaan arah kompas

Gambar 4.4 Praktek pembacaan kompas

Gambar 4.5 Grafik *error* arah kompas

4.1.4 Pengujian Kontrol pada Navigasi GPS dan kompas

Pengujian kontrol navigasi GPS secara parsial dilaksanakan dengan terlebih dahulu menginputkan *waypoint* tujuan (*latitude longitude*). Setelah signal GPS valid dan jumlah satelit yang tertangkap lebih dari 4 maka kontroler akan menjalankan kontrol navigasi otomatis dengan mengarah *waypoint* yang akan dituju. Output kontrol dari navigasi GPS diaktualisasikan terhadap perubahan sudut aileron. Kontrol navigasi GPS menggunakan kontrol proporsional dan derivatif. Dengan PID 1490 posisi aileron netral pesawat terbang lurus ke arah tujuan. Berikut hasil pengujian pada kontrol navigasi GPS di darat:

Tabel 4.5 Navigasi GPS dan Kompas

Gambar 4.6 Grafik *waypoint* dan ketinggian

Pada pengujian parsial tersebut dapat dilihat bahwa arah heading bernilai 300. Hal ini dikarenakan pesawat menuju ke arah barat laut. Output dari kontrol 1400 sehingga pesawat terbang lurus ke barat mengikuti *waypoint* yang telah dimasukkan ke program. Hasil pengujian yang ditampilkan yaitu *heading compass*, *altitude*, *speeed*, *waypoint*, output kontrol navigasi.

4.2 Pengujian Integrasi

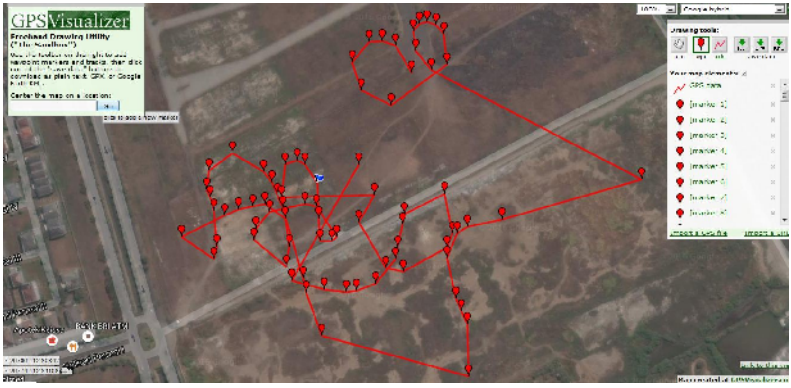
Pengujian integrasi merupakan pengujian secara keseluruhan dari semua fungsi pengujian parsial sebelumnya secara nyata dengan kondisi terbang. Pengujian yang dilaksanakan meliputi pengujian kontrol manual dan pengujian kontrol otomatis kestabilan pesawat di udara dan pengujian integrasi dari sistem kestabilan dan navigasi GPS *waypoint.* Sistem kontrol menggunakan catu daya yang terpisah dengan catu daya motor DC *brushless.* Hal tersebut dimaksudkan untuk menghindari drop tegangan dan arus pada saat terjadi indikasi *low battery* sehingga tidak mengganggu sistem kontrol pesawat. Nilai Kp dan Ki kontrol stabilitas pesawat serta nilai Kp dan Kd kontrol navigasi didapat memlaui metode *trial and error* pengujian di lapangan dengan melihat respon sistem secara langsung.

Tabel 4.6 Pengujian pencapaian target ke *waypoint*

Gambar 4.7 Grafik pencapaian ke *waypoint*

4.2.1 Pengujian Kontrol GPS tanpa kompas

Pada pengujian kontrol GPS tanpa kompas di udara, pesawat tidak bisa mencapai ke *waypoint* tujuan dengan tepat pesawat berputar putar dahulu untuk menemukan titik tujuan yang tepat dan selanjutnya akan menuju ke titik berikutnya begitu seterusnya, dengan kondisi tersebut maka pesawat akan mustahil untuk melakukan mendarat otomatis secara sempurna karena pesawat tidak bisa terbang lurus ke arah tujuan, berikut gambar plot percobaan di udara kontrol GPS tanpa kompas:



Gambar 4.8 Plot data GPS tanpa kompas

4.2.2 Pengujian Kontrol Pendaratan Otomatis dengan GPS dan Kompas *Waypoint*

Pada pengujian sistem navigasi otomatis dengan *waypoint* GPS di udara dilaksanakan dengan empat waypoint, yaitu *waypoint* tujuan dan *home waypoint* yang letaknya lurus sejajar agar pesawat bisa melaksanakan ancang ancang pendaratan yang tepat dan baik . Pertama-tama pesawat diterbangkan dengan mode manual. Setelah mencapai ketinggian yang aman dan arah ke daerah pendratan maka mode otomatis diaktifkan dan selanjutnya plant akan bergerak mencari arah ke *waypoint* tujuan. Setelah jarak antara plant dan *waypoint* tujuan kurang dari 50 meter maka output kontrol navigasi dinon-aktifkan. Pada saat itu pesawat hanya menjalankan kontrol kestabilan di udara sampai jarak dengan *waypoint*1 lebih dari 50 meter dan selanjutnya akan mencari arah menuju *home waypoint* setiap titik tujuan mempunyai ketinggian yang berbeda beda agar pesawat semakin lama semakin turun ketinggiannya.

Pada pengujian kontrol tersebut pesawat mengalami kesulitan dalam hal kecepatan dikarnakan faktor kecepatan angin dan udara panas yang naik dari permukaan bumi.

Untuk mengantisipasi permasalahan tersebut maka untuk kecepatan motor dikontrol secara manual agar antara pergerakan pesawat dan kecepatan setabil sesuai yang digarapkan, karena kesalahan sedikit mengakibatkan pesawat jatuh dan hancur.

Pada pengujian pendaratan otomatis didapatkan hasil bahwa pesawat dapat menuju waypoint tujuan dan setelah sampai di waypoint tujuan pesawat dapat mengurangi ketinggian sampai *home waypoint*. Data *latitude*, *longitude* dan ketinggian pesawat pada saat terbang selanjutnya diplot kedalam peta satelit sehingga didapatkan hasil plot peta seperti pada gambar 4.10.

Gambar 4.9 Plot data GPS mengunakan kompas

4.2.3 Pengujian Daya Tahan Baterai

Pengujian tambahan yang dilaksanakan pada sistem ini adalah pengujian daya tahan baterai litium-polimer 3 *cell* 2200mah pada saat terbang di udara. Pengujian tersebut dilakukan dengan menambahkan rangkaian pembagi tegangan sebelum masuk ke pin ADC(*Analog to Digital Coverter*). ADC yang digunakan adalah adc 10 bit (1023) dengan tegangan referensi yang digunakan adalah sebesar 5 VDC. Tegangan awal baterai sebelum terbang adalah sebesar 12,45 volt. Batas bawah baterai yang digunakan pada saat terbang adalah sebesar 11,1 volt. Data hasil pembacaan tegangan baterai selanjutnya dikirimkan melalui perangkat telemetri dan diterima oleh komputer. Grafik tegangan baterai pada pengujian terbang dapat dilihat pada gambar 4.10.

Gambar 4.10 Grafik hasil pengujian tegangan baterai saat terbang

4.3 Analisa Pengujian

Pengujian aksi kontrol terhadap *plant* dilaksanakan untuk mengetahui kesesuaian antara perancangan dengan hasilnya. Terdapat beberapa perbedaan antara hasil dengan teori yang diharapkan. Berikut analisa yang diperoleh dari perbandingan hasil pengujian dengan perancangan pada bab sebelumnya antara lain:

1. Hasil pengujian kontrol manual melaui mikrokontroler di lapangan didapatkan hasil bahwa respon servo aktuator sedikit memiliki delay jika dibandingkan dengan respon servo tanpa melalui mikrokontroler tetapi delay tersebut masih dalam batas toleransi wajar. Pada pengujian saat catu daya sistem kontrol menjadi satu dengan catu daya motor *brushless* mengakibatkan sistem kontrol terganggu saat tegangan battery kurang dari 11 volt.

2. Pengujian sistem pergerakan motor servo sistem kendali pada pesawat untuk sudut *pitch* dan *roll* harus dibatasi dari batas maksimal 45^{o} ke atas dan ke bawah, karena akan mengakibatkan kemacetan atau gangguan motor servo saat di udara. .

3. Hasil pengujian pada kontrol ketinggian didapatkan bahwa penerimaan jarak ketinggian dibawah 5 meter dengan menggunakan *barometer* kurang bekerja dengan baik, karena

dengan tempat dan daerah yang berbeda ketinggian di atas permukaan laut berbeda juga, sehingga menggunakan sensor *ultrasonic* untuk bekerja di ketinggian rendah sehingga akan menghasilkan ketinggian yang presisi di tempat dan daerah yang berbeda.

4. Kontrol PID *waypoint* berfungsi sesuai dengan perancangan yang diharapkan. Hasil pengujian yaitu bahwa pesawat dapat menuju ke *waypoint* tujuan selanjutnya kembali ke *home waypoint*. Dengan baik dengan menambahkan kompas, karena tanpa kompas pesawat menuju ke *waypoint* tujuan tidak bisa tepat dan cepat.

5. Sistem kontrol kestabilan pesawat di udara sangat dipengaruhi oleh proses kalibrasi sensor IMU pada saat sebelum terbang. Jika proses kalibrasi dilakukan pada sudut yang salah maka sistem akan sulit untuk mendapatkan *steady state* nya. Selain itu kondisi aerodinamis pesawat yang kurang standar (pemasangan motor *brushless*, *wing incident* dan CG pesawat yang kurang tepat, berat pesawat yang tidak sesuai dengan daya angkat motor) mengakibatkan sistem menjadi lebih sulit untuk mendapatkan kestabilan.

[Halaman ini sengaja dikosongkan]

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan data hasil uji terbang sistem kontrol pendaratan otomatis baik navigasi GPS *waypoint* tanpa kompas dan menggunakan kompas pada pergerakan UAV ini diperoleh beberapa kesimpulan antara lain sebagai berikut:

1. Kontrol manual dan otomatis pada UAV menggunakan modul mikrokontroler Arduino Mega 2560 dapat dilaksanakan menggunakan 3 *channel* (aileron, elevator, dan saklar) sedangkan 2 *channel*(*throttle* dan rudder) menggunakan *receiver* radio kontrol.
2. Kontrol sistem navigasi di udara tanpa menggunakan kompas akan mengakibatkan pesawat sulit untuk menemukan koordinat *waypoint* tujuan.
3. Pengujian terbang kontrol pendaratan otomatis kestabilan pesawat di udara dapat dilaksanakan dengan baik dengan menambahkan kompas yang digabungkan dengan *accelerometer* sehingga kompas tetep bekerja meskipun pesawat dalam kondisi miring.
4. Pengujian kontrol ketinggian otomatis pesawat di udara dapat dilaksanakan dengan baik dengan ketinggian dibawah 5 meter dengan menggunakan sensor *ultrasonic*..
5. Pengujian terbang dengan pendaratan otomatis dengan *waypoint* berhasil dengan syarat-syarat seperti tingkat standarisasi aerodinamika pesawat, faktor angin dan keakuratan GPS yang dilaksanakan pada saat sebelum terbang.

5.2 Saran

Ada beberapa saran yang terkait dengan kendala dan kekurangan yang dihadapi dalam penyusunan tugas akhir ini sehingga diharapkan menjadi bahan pertimbangan pada tahap pengembangan selanjutnya yang antara lain sebagai berikut:

1. Perancangan *hardware* UAV harus memperhatikan bentuk aerodinamis pesawat sehingga pergerakan pesawat benar-benar stabil. Selain itu perancangan *hardware* harus memperhatikan

faktor berat terhadap daya dorong maksimum dari motor *brushless.*

2. Penambahan program *interface* di darat untuk mempermudah monitoring sistem kontrol dan memasukkan GPS waypoint dengan mudah
3. Penyiapan *training area* di Institut Teknologi Sepuluh Nopember untuk mendukung pengujian terbang pesawat baik model *fixwing* dan *rotary wing* untuk penelitian-penelitian selanjutnya.

Demikian saran yang dapat disampaikan dengan harapan dapat dijadikan seagai bahan pertimbangan sehingga dapat bermanfaat untuk pengembangan selanjutnya.

DAFTAR PUSTAKA

[1] PORDIRGA AEROMODELLING PB FASI, "Buku Panduan Aeromodelling Indonesia", Jakarta, 2014

[2] Elizabeth Bone. "*Unmanned Aerial Vehicles: Background and Issues for Congress*". Congressional Research Service, Washington, DC. 2003.

[3] GNU, Libraries and tutorials for Arduino IMU processing, diakses pada tanggal 1 Pebruari 2016, http://www.tkjelectronics.com/

[4] HaiYang Chao, YongCan Cao and YangQuan Chen," *Autoplots for Small Unmanned Aerial Vehicles: A Survey"*, 2010

[5] Ippolito Corey, *"An Autonomous Autopilot Control System Design for Small-Scale UAV's"*, QSS Group, Inc. NASA Ames Research Center, 2012

[6] Ogata, katsuhiko, Edi Laksono (Penterjemah). 1993. Teknik Kontrol Automatik (Sistem Pengaturan) Jilid 2. Jakarta : Erlangga

[7] Antono Djodi, "Motor DC Brushless Tiga Fase-Satu Kutub", Teknik Elektro Politeknik Negeri Semarang, Semarang, 2012

[8] *Understanding PWM*, diakses pada tanggal 12 Maret 2016, http://ebldc.com/?p=48

[9] Geoffrey Blewitt, "*Basics of the GPS Technique: Observation Equations"*, Department of Geomatics, University of Newcastle Newcastle upon Tyne, NE1 7RU, United Kingdom, 1997.

[10] Dasar-Dasar Autopilot atau *Flight Controller*,diakses pada tanggal 7 September 2016, http://aeroengineering.co.id/.

[11] Pesawat Tanpa Awak, *Unmanned Aerial Vehicle* (UAV), diakses tanggal 8 September 2016, http://zonaelektro.net/.

[12] Products Engines for Various Size Tactical UAVs, diakses tanggal 9 September 2016, http://www.uavengines ltd.co.uk/products/.

[13] Theodore A. Talay,"*Introduction to the Aerodynamics of Flight*", National Aeronautics and Space Administration, Washington, D.C. 1975.

[14] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[15] https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf

[16] http://www.pishrobot.com/files/products/datasheets/srf02.pdf

[17] http://www.alldatasheet.com/datasheet-pdf/pdf/131940/FREESCALE/MPXV7002DP.html
[18] https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf
[19] http://download.ardupilot.org/downloads/wiki/advanced_user_tools/PPM -Encoder-V3-Manual.pdf

LAMPIRAN A

Lembar Pengesahan Proposal

Jurusan Teknik Elektro
Fakultas Teknologi Industri - ITS

TE 141599 TUGAS AKHIR – 4 SKS

Nama Mahasiswa	:	Erwan Aprilian
Nomer Pokok	:	2214 105 012
Bidang Studi	:	Elektronika
Tugas Diberikan	:	Semester Genap 2016/2017
Dosen Pembimbing	:	1. Ronny Mardiyanto, ST., MT.,Ph.D.

15 SEP 2016

Judul Tugas Akhir : Pengembangan Sistem Pendaratan Otomatis pada Pesawat Tanpa Awak.
Development of Autolanding Sistem for Unmanned Aerial Vehicle (UAV).

Uraian Tugas Akhir :
UAV adalah salah satu wahana tanpa awak di udara yang dapat terbang tanpa pilot. Secara teknis kemampuan terbang dan sistem kerja pesawat ini menyerupai kondisi pesawat sebenarnya. Selanjutnya UAV tersebut dapat digunakan untuk keperluan baik dilingkup militer maupun sipil. Pada proposal tugas akhir ini akan dirancang dan direalisasikan pengembangan sistem *autopilot* pada UAV dengan tambahan mendarat otomatis. Sistem ini menggunakan kontrol manual dan *autopilot*. Pada mode manual, pengguna secara manual mengendalikan pergerakan pesawat melalui RC sedangkan pada mode *autopilot* pesawat dikendalikan oleh mikrokontroler yang mengolah data-data sensor untuk mengarahkan pesawat dari satu titik ke titik yang lain. Sensor yang digunakan antara lain IMU (*Inertial Measurement Unit* yang didalamnya terdapat *gyroscope* dan *accelerometer*), GPS, *barometric altimeter*, *airspeed* dan *ultrasonic*. Mikrokontroler menerima data input dari sensor secara terus-menerus, mengolah sampel data dan menghasilkan output untuk menggerakkan aktuator berupa servo. Pengolahan data sensor menggunakan kontrol PID (*Proportional Integral Derivative*) dengan metode Ziegler Nichols. Manuver kontrol *autopilot* meliputi *roll* (sumbu x), *yaw* (sumbu y), *pitch* (sumbu z), ketinggian dan *heading* pesawat. Pesawat akan terkoneksi dengan *ground station* melalui perangkat telemetri. Sistem navigasi ini diharapkan dapat secara tepat mengarahkan pesawat menuju satu titik atau lebih dengan toleransi kesalahan < 10 meter pada ketinggian 90-100 meter. Selain itu pesawat dapat terbang dengan radius 1-2 km dari *ground station*.

Jurusan Teknik Elektro
Fakultas Teknologi Industri - ITS

Dosen Pembimbing,

Ronny Mardiyanto, ST., MT.,Ph.D.
Nip : 198101182003121003

Mengetahui,
Koordinator Program Studi S1

Dedet Chandra Riawan,ST.M.Eng.Ph.D.
Nip : 197311192000031001

Menyetujui,
Kepala Laboratorium AJ 304

Ir. Tasripan, MT.
Nip : 196204181990031004

[Halaman ini sengaja dikosongkan]

LAMPIRAN B

A.1 *Listing* Program Arduino pada Program Utama

```
#include "Arduino.h"
#include <ChibiOS_AVR.h>
#include <Servo.h>
#include <eRCaGuy_Timer2_Counter.h>
#include "RC_PPM.h"
#include "GPS.h"
#include "myAlgorithm.h"
#include "myServo.h"
#include "mySensor.h"

const uint8_t LED_PIN = 13;
volatile double heading = 0;

//--------------------------------------------------------------------------------

//(Thread baca ultrasonic)
static THD_WORKING_AREA(waThreadUS, 250);
static THD_FUNCTION(ThreadUS ,arg) {
 systime_t time = chVTGetSystemTimeX(); // T0
  while (1) {
        time += MS2ST(100);
        get_SRF02();
        chThdYield(); //(
        chThdSleepUntil(time);
  }
}

//(Thread baca compass & pressure)
static THD_WORKING_AREA(waThreadCMPS_PRES, 250);
static THD_FUNCTION(ThreadCMPS_PRES ,arg) {
 systime_t time = chVTGetSystemTimeX(); // T0
  while (1) {
        time += MS2ST(100);
        set_Heading(calc_and_get_heading());
```

```
      if(get_current_Altitude_BMP280() <= 5){ //( ketika kurang
    dari 5m pakai ultrasonic)
      set_Altitude(get_UltraSonic_Distance());
      }
      else{
      set_Altitude(get_current_Altitude_BMP280()); //(Ketinggian
    ultrasonic diatas 5m pakai barometer)
      }
      chThdSleepUntil(time); //(Berhenti sementara untuk
    mempersilahkan program lain jalan)
  }
}

//(Thread baca PPM & GPS)
static THD_WORKING_AREA(waThreadPPMandGPS, 164);
static THD_FUNCTION(ThreadPPMandGPS ,arg) {
unsigned int reading = 0;
  while (1) {

      if(get_Channel_value(5) >= 1600){ //(Bila chanel 5 on maka
    mode otomatis)
      set_MODE(AUTO_MODE);
      }
      else{
      set_MODE(MANUAL_MODE);
      }

      if(get_MODE() == AUTO_MODE){
   int roll_filtered=(get_Servo_Roll_Value()-1500)/2+1500;
   int pitch_filtered=(get_Servo_Pitch_Value()-1500)/2+1500;
   set_Servo_Roll_us(roll_filtered);
   set_Servo_Pitch_us(pitch_filtered);
      update_waypoint();   // (update waypoint)
// (Ketika ketinggian 3 meter pesawat mempertahankan sikap level)
  if(get_Altitude()<=3.0){
    if((get_Servo_Roll_Value()>1400 && get_Servo_Roll_Value()<
      1600) && (get_Servo_Pitch_Value()>1400 &&
      get_Servo_Pitch_Value()< 1600)){
     }
```

```
    }
      }
      else if(get_MODE() == MANUAL_MODE){
   int roll_filtered=(get_Channel_value(1)-1500)/2+1500;
   int pitch_filtered=(get_Channel_value(2)-1500)/2+1500;
      set_Servo_Roll_us(roll_filtered);
      set_Servo_Pitch_us(pitch_filtered);
      }

      update_GPS();          // (baca GPS)
      //(Baca Ultrasonic)
      if(Serial1.available()>=2){
      reading = Serial1.read()<<8;//receive high byte (overwrites
      previous reading) and shift high byte to be high 8 bits
      reading |= Serial1.read(); // receive low byte as lower 8 bits
      Serial1.flush();
      set_UltraSonic_Distance((double)(reading/100.0f));//(Dibagi 100
      agar mendapat satuan meter)
      }
      chThdYield(); //(mengijinkan untuk thread lain untuk jalan)
  }
}
//------------------------------------------------------------------------------
// thread 2 - print main thread count every second
// 100 byte stack beyond task switch and interrupt needs
static THD_WORKING_AREA(waThreadPrintData, 120);
static THD_FUNCTION(ThreadPrintData, arg) {

  // (menampilkan setiap detik)
  while (1) {
 // (Tidur untuk satu mili secon)
    chThdSleepMilliseconds(250);
 // (Menampilkan untuk telemetry)
  Serial2.print(F(" Yaw: "));
  Serial2.print(get_Heading());
  Serial2.print(" Jarak:");
  Serial2.print(get_Jarak_ke_Tujuan());
  Serial2.print("  Altitude: ");
  Serial2.print(get_Altitude());
```

```
    Serial2.print(" MODE:");
    Serial2.print(get_MODE());
    Serial2.print(" US:");
    Serial2.print(get_UltraSonic_Distance());
    Serial2.print(" Spd:");
    Serial2.print(get_mpx_KMPH());
    Serial2.print(" LAT,LNG:");
    Serial2.print(get_GPS_lat(), 6);
    Serial2.print(",");
    Serial2.print(get_GPS_lng(), 6);
   Serial2.print(" PID:");
   Serial2.print(get_PID_Output());
   Serial2.print(" Err Roll:");
   Serial2.print(get_error_roll());
   Serial2.print(" Err Pich:");
   Serial2.print(get_error_pitch());
   Serial2.print(" Roll:")
   Serial2.print(get_Servo_Roll_Value());
   Serial2.print(" Pich:")
   Serial2.print(get_Servo_Pitch_Value():
    for ( byte x = 1; x <= 8; x++)// ch 1 sd 8
        {
        Serial.print("  Ch.");
        Serial.print(x);
        Serial.print(": ");
        Serial.print(get_Channel_value(x));     // (Pengelompokan chanel)
        Serial.print(" ");
        }
        Serial2.println();
    chThdYield();//(mengijinkan untuk thread lain untuk jalan)
  }
}
//------------------------------------------------------------------------------

void setup() {
        timer2.setup();
Serial.begin(115200);
        Serial2.begin(57600); //(Boudrate untuk telemetry)
        if (CH_CFG_TIME_QUANTUM != 0) {
```

```
        Serial.println("Untuk mengaktifkan mode COOPERATIVE maka
        CH_CFG_TIME_QUANTUM harus di set 0");
        while(1);
        }
        init_MPXV7002dp();
        init_SRF02();
        init_BMP280();
        init_MPU6050();
        init_HMC5883L();
        init_servo();
        init_GPS();
        init_waypoint();
        init_RC_PPM();
        delay_T2(200);
        chBegin(mainThread);
        // (chanel akan lanjut dan tidak kembali ke awal, berjalan
        sesuai urutannya)

        while(1) {}
}
//------------------------------------------------------------------------
// (Berjalan sesuai urutan prioritas normal)
void mainThread() {
// (memulai menampilkan urutan)
  chThdCreateStatic(waThreadUS, sizeof(waThreadUS), // (ultrasonic)
                                                    NORMALPRIO+7          ,
        ThreadUS, NULL);
// (start baca compass dan baro thread)
chThdCreateStatic(waThreadCMPS_PRES,
        sizeof(waThreadCMPS_PRES),
                                                    NORMALPRIO+6,
        ThreadCMPS_PRES, NULL);
// (start print thread)
  chThdCreateStatic(waThreadPrintData,
        sizeof(waThreadPrintData),//serial
                    NORMALPRIO +5, ThreadPrintData, NULL);
// (start print thread)
  chThdCreateStatic(waThreadPPMandGPS,
        sizeof(waThreadPPMandGPS), //(gps)
```

```
                NORMALPRIO+4 , ThreadPPMandGPS, NULL);


// (increment counter)
  while (1) {
  }
}
//--------------------------------------------------------------------------
void loop() {
//(Tidak dipakai)
}
```

A.2 Listing Program Arduino pada Library GPS.cpp dan GPS.h

----------GPS.cpp----------

```
#include "GPS.h"
// radius of earth in meters
#define RADIUS_OF_EARTH 6378100
// scaling factor from 1e-7 degrees to meters at equater
// == 1.0e-7 * DEG_TO_RAD * RADIUS_OF_EARTH
#define LOCATION_SCALING_FACTOR 0.011131884502145034f
// inverse of LOCATION_SCALING_FACTOR
#define LOCATION_SCALING_FACTOR_INV 89.83204953368922f

static const uint32_t GPSBaud = 9600;
// The TinyGPS++ object
TinyGPSPlus gps;

//(lokasi awal dan pulang)
double Home_Start_LAT;
double Home_Start_LON;

// (Lokasi waypoint sebelumnya. Digunakan untuk jalan dan
    kalkulasi ketinggian)
Location prev_WP_loc {};

// (Lokasi jalannya pesawat)
struct Location current_loc {};
```

```
// (lokasi waypoint. ketinggian, mengikuti track).
Location next_WP_loc {};

void init_GPS(void){
      Serial3.begin(GPSBaud);       //(gps menggunakan serial3)
      Wait_Gps_Data();
}

void Wait_Gps_Data(void) { //(Menunggu GPS mendapatkan satelit)
Serial.print("Waiting GPS Data");
      uint8_t dot = 0;
//(akan bekerja jika GPS menangkap 4 satelit atau lebih)
while (!gps.location.isValid() && gps.satellites.value() < 4) {
            if (dot > 15) {//15
            Serial.print(gps.satellites.value());
            Serial.println();
            dot = 0;
            }
            Serial.print(".");
            dot++;
            update_GPS();
            smartDelay(200); //(Tertampil dalm waktu 200 milisecon)
      }
      //(Menentukan lokasi awal untuk pulang dengan cara
      inisialisasi awal pada saat mendapat satelit yang diinginkan)
      Home_Start_LAT = gps.location.lat();
      Home_Start_LON = gps.location.lng();
      Serial.println("GPS Pos Valid");
      delay_T2(50); //(tampil waktu 50 milisecon)
}
void smartDelay(unsigned long ms)
{
      unsigned long start = timer2.get_micros()*1000;
      do{
          update_GPS();
      } while ((timer2.get_micros()*1000) - start < ms);
}

void update_GPS(){
```

```
        if (Serial3.available() > 0){
           gps.encode(Serial3.read());
        }
}

double get_GPS_lat(){
        if (gps.location.isValid()){
            return gps.location.lat();
        }
        else return 0;
}

double get_GPS_lng(){
        if (gps.location.isValid()){
            return gps.location.lng();
        }
        else return 0;
}

double get_HOME_LAT(){
        return Home_Start_LAT;
}
double get_HOME_LNG(){
        return Home_Start_LON;
}

float get_distance(const struct Location &loc1, const struct Location
        &loc2){
        return gps.distanceBetween(loc1.lat, loc1.lng, loc2.lat, loc2.lng);
}

float courseTo(const struct Location &loc1, const struct Location
        &loc2){
        return gps.courseTo(loc1.lat, loc1.lng, loc2.lat, loc2.lng);
}

float longitude_scale(const struct Location &loc)
{
```

```
    float scale = cosf(loc.lat * 1.0e-7f * DEG_TO_RAD); //(Merubah
    drajat ke Radian)
    return constrain_float(scale, 0.01f, 1.0f);
}

// (sudut kembali untuk 2 koordinat)
int32_t get_bearing_cd(const struct Location &loc1, const struct
    Location &loc2)
{
  int32_t off_x = loc2.lng - loc1.lng;
  int32_t off_y = (loc2.lat - loc1.lat) / longitude_scale(loc2);
  int32_t bearing = 9000 + atan2f(-off_y, off_x) * 5729.57795f;
  if (bearing < 0) bearing += 36000;
  return bearing;
}
//(Mencapai target waypoin dari 2 titik point)
bool location_passed_point(const struct Location &location,
                    const struct Location &point1,
                    const struct Location &point2)
{
  return location_path_proportion(location, point1, point2) >= 1.0f;
}


float location_path_proportion(const struct Location &location,
                      const struct Location &point1,
                      const struct Location &point2)
{
  Vector2f vec1 = location_diff(point1, point2);
  Vector2f vec2 = location_diff(point1, location);
  float dsquared = sq(vec1.x) + sq(vec1.y);
  if (dsquared < 0.001f) {
    // the two points are very close together
    return 1.0f;
  }
  return (vec1 * vec2) / dsquared;
}
```

```
Vector2f location_diff(const struct Location &loc1, const struct
    Location &loc2)
{
  return Vector2f((loc2.lat - loc1.lat) *
    LOCATION_SCALING_FACTOR,
        (loc2.lng - loc1.lng) * LOCATION_SCALING_FACTOR
    * longitude_scale(loc1));
}

// (Pembatasan nilai)
float constrain_float(float amt, float low, float high)
{
    if (isnan(amt)) {
        return (low+high)*0.5f;
    }
    return ((amt)<(low)?(low):((amt)>(high)?(high):(amt)));
}

----------------------------------GPS.h-----------------------------------------
#ifndef GPS_H_
#define GPS_H_

#include <TinyGPS++.h>
#include <stdlib.h>
#include <math.h>
#include "myAlgorithm.h"
#include "vector2.h"

// used to pack structures
#if defined(__AVR_ATmega1280__) ||
    defined(__AVR_ATmega2560__)
#define PACKED
#else
#define PACKED __attribute__((__packed__))
#endif

struct PACKED Location_Option_Flags {
  uint8_t relative_alt : 1;         // 1 if altitude is relateive to home
```

```
    uint8_t unused1         : 1;                // unused flag (defined so that
        loiter_ccw uses the correct bit)
    uint8_t loiter_ccw   : 1;          // 0 if clockwise, 1 if counter clockwise
    uint8_t terrain_alt  : 1;          // this altitude is above terrain
};

struct PACKED Location {
    union {
        Location_Option_Flags flags;                    ///< options bitmask
        (1<<0 = relative altitude)
        uint8_t options;                            /// allows writing all flags to
        eeprom as one byte
    };
    // by making alt 24 bit we can make p1 in a command 16 bit,
    // allowing an accurate angle in centi-degrees. This keeps the
    // storage cost per mission item at 15 bytes, and allows mission
    // altitudes of up to +/- 83km
    int32_t alt:24;                                 ///< param 2 - Altitude in
        centimeters (meters * 100)
    int32_t lat;                                ///< param 3 - Lattitude * 10**7
    int32_t lng;                                 ///< param 4 - Longitude * 10**7
};

// flight mode specific
struct {
    // Flag for using gps ground course instead of INS yaw.  Set false
        when takeoff command in process.
    bool takeoff_complete:1;

    // Flag to indicate if we have landed.
    // Set land_complete if we are within 2 seconds distance or within 3
        meters altitude of touchdown
    bool land_complete:1;

    // should we fly inverted?
    bool inverted_flight:1;

    // should we disable cross-tracking for the next waypoint?
    bool next_wp_no_crosstrack:1;
```

```
// should we use cross-tracking for this waypoint?
bool no_crosstrack:1;

// in FBWA taildragger takeoff mode
bool fbwa_tdrag_takeoff_mode:1;

// have we checked for an auto-land?
bool checked_for_autoland:1;

// denotes if a go-around has been commanded for landing
bool commanded_go_around:1;

// Altitude threshold to complete a takeoff command in autonomous
    modes.  Centimeters
// are we in idle mode? used for balloon launch to stop servo
// movement until altitude is reached
bool idle_mode:1;

// used to 'wiggle' servos in idle mode to prevent them freezing
// at high altitudes
uint8_t idle_wiggle_stage;

// Altitude threshold to complete a takeoff command in autonomous
// modes.  Centimeters above home
int32_t takeoff_altitude_rel_cm;

// Minimum pitch to hold during takeoff command execution.
    Hundredths of a degree
int16_t takeoff_pitch_cd;

// the highest airspeed we have reached since entering AUTO. Used
// to control ground takeoff
float highest_airspeed;

// initial pitch. Used to detect if nose is rising in a tail dragger
int16_t initial_pitch_cd;

// turn angle for next leg of mission
```

```
    float next_turn_angle {90};

    // filtered sink rate for landing
    float sink_rate;

    // time when we first pass min GPS speed on takeoff
    uint32_t takeoff_speed_time_ms;

    // distance to next waypoint
    float wp_distance;

    // proportion to next waypoint
    float wp_proportion;

    // last time is_flying() returned true in milliseconds
    uint32_t last_flying_ms;

    // once landed, post some landing statistics to the GCS
    bool post_landing_stats;
} auto_state;



void init_GPS(void);
void Wait_Gps_Data(void);
void update_GPS(void);
void smartDelay(unsigned long ms);
void update_GPS();

double get_GPS_lat();
double get_GPS_lng();

double get_HOME_LAT();
double get_HOME_LNG();

float get_distance(const struct Location &loc1, const struct Location
      &loc2);
float courseTo(const struct Location &loc1, const struct Location
      &loc2);
```

```
float longitude_scale(double lat2, double long2);
int32_t get_bearing_cd(double lat1, double long1, double lat2, double
    long2);

bool location_passed_point(const struct Location &location,
                    const struct Location &point1,
                    const struct Location &point2);
float location_path_proportion(const struct Location &location,
                      const struct Location &point1,
                      const struct Location &point2);
Vector2f location_diff(const struct Location &loc1, const struct
    Location &loc2);
float constrain_float(float amt, float low, float high) ;

#endif /* GPS_H_ */
```

A.3 Listing Program Arduino pada Library RC_PPM.ccp dan RC_PPM.h

```
-------------------------------------RC_PPM.ccp-------------------------------
    -
#include "RC_PPM.h"


const int NumberOfChannels = 8; // (Banyaknya Ch PWM yang
    dikeluarkan)
const byte InputPin = 2; // (Input pada pin 2 pada Arduino
    Mega2560)
const int FrameSpace = 3000;   // (waktu 8 ms)


volatile long Current_Time;
volatile long Last_Spike;

volatile byte Current_Channel = 0;
volatile int Spike_Length[NumberOfChannels + 1];
volatile int LastChannelFlag = false;


void init_RC_PPM(void){
```

```
        pinMode(InputPin, INPUT);
        delay_T2(100);
        attachInterrupt( digitalPinToInterrupt(InputPin), Spike, RISING);
        Last_Spike = timer2.get_micros();
}

void Falling(void)
{
      Current_Time = timer2.get_micros();
      Spike_Length[Current_Channel] = Current_Time - Last_Spike;
      Current_Channel = 0;
      Last_Spike = Current_Time;
      LastChannelFlag = false;
      detachInterrupt( digitalPinToInterrupt(InputPin));
      attachInterrupt( digitalPinToInterrupt(InputPin), Spike, RISING);
}

//(Penyimpanan panjang spike)
void Spike(void)
{
 Current_Time = timer2.get_micros();
 Spike_Length[Current_Channel] = Current_Time - Last_Spike;
 if (Spike_Length[Current_Channel] > FrameSpace) {
       Current_Channel = 0;       // oops, you were actually in the frame
      space --- need to add correction
 }
 Last_Spike = Current_Time;                    // Set the current time as
      the previous time to the next one.

 Current_Channel = Current_Channel + 1;              // Reading the next
      channel now

 if (Current_Channel > NumberOfChannels)                                  //
      Special case. Must wait for it to fall manually.
 {
  LastChannelFlag = true;
  detachInterrupt( digitalPinToInterrupt(InputPin));
  attachInterrupt( digitalPinToInterrupt(InputPin), Falling, FALLING);
 }
```

```
}

bool get_LastChannelFlag(void){
	return LastChannelFlag;
}

void LastChannel(void)
{
  while(digitalRead(InputPin) == HIGH);
  Current_Time = timer2.get_micros();  // Now it has fallen
  Spike_Length[Current_Channel] = Current_Time - Last_Spike;
  Current_Channel = 0;
  Last_Spike = Current_Time;
  LastChannelFlag = false;
}

int get_Channel_value(int channel){
	return  (constrain(Spike_Length[channel],  1200,  1800));		//
	Mapping values may need to be changed depending on receiver
}
```

----------RC_PPM.h----------

```
#ifndef RC_PPM_H_
#define RC_PPM_H_
#include <Arduino.h>
#include <ChibiOS_AVR.h>
#include <eRCaGuy_Timer2_Counter.h>
#include "myAlgorithm.h"

void init_RC_PPM(void);
void Falling(void);
void Spike(void);
void LastChannel(void);
int get_Channel_value(int channel);
bool get_LastChannelFlag(void);

#endif /* RC_PPM_H_ */
```

A.5 Listing Program Arduino pada Library Myalgorithm.cpp dan Myalgorithm.h

```
-------------------------------------Myalgorithm.cpp--------------------------
#include "myAlgorithm.h"

volatile uint8_t Next_Wp = 0;
//(koordinat dan ketinggian waypoint yang dimasukkan manual sebagai misi penerbangan)
#define JUMLAH_WAYPOINT     5
double waypoint_data[JUMLAH_WAYPOINT][3] =
          //Latitude, Longitude, Altitude(meter)
          { -7.282795,112.812767, 0},          // (home location nilai berapa saja ga masalah)
          { -7.277915,112.798227, 5}
          { -7.277772,112.797993, 10},
          { -7.277755,112.797727, 30},
          { -7.277647,112.797508, 40}
};

double Jarak_ke_Tujuan;  //(meter)
double Sudut_ke_Tujuan;  //(derajat)
const float waypoint_radius = 50.0f;          //(Pada saat jarak 50 m dari waypoint tujuan dianggap mencapai)

volatile uint8_t MODE;

#define  FIND_HEADING_TARGET 0
#define NAVIGATE_TO_TARGET 1
uint8_t STATE = FIND_HEADING_TARGET;

//(Nilai PID parameter Roll untuk berbelok)
int error_roll, last_error_roll;
double Setpoint_roll, PID_Output_roll;
double Kp_roll = 1, Ki_roll = 0, Kd_roll = 0.1; //ch1
volatile unsigned long lastTime_roll;
double errSum_roll;
double Roll_Servo;

//(Nilai PID parameter Pitch untuk atur ketinggian)
```

```
int error_pitch, last_error_pitch;
double Setpoint_pitch, PID_Output_pitch;
double Kp_pitch = 3, Ki_pitch = 0.00, Kd_pitch = 0.1; //ch2
volatile unsigned long lastTime_pitch;
double errSum_pitch;
double Pitch_Servo;


void set_MODE(uint8_t mode){
      MODE = mode;
}

uint8_t get_MODE(void){
      return MODE;
}

void delay_T2(unsigned long time){
      unsigned long target_time = timer2.get_micros()+(time*1000);
      while(timer2.get_micros() <= target_time);
}
void init_waypoint(void){
      //(Pengaturan waypoint 0 sebagai HOME)
      waypoint_data[0][0] = get_HOME_LAT();
      waypoint_data[0][1] = get_HOME_LNG();
      waypoint_data[0][2] = 0;//20.0

      Next_Wp = 1;          //(waypoint 0 adalah home location)
}

void update_waypoint(void){
      struct Location loc1, loc2;
      loc1.lat = get_GPS_lat();
      loc1.lng = get_GPS_lng();
      loc1.alt = get_Altitude();
      loc2.lat = waypoint_data[Next_Wp][0]; //(Tempat Latitude)
      loc2.lng = waypoint_data[Next_Wp][1]; //(Tempat Longitude)
      loc2.alt = waypoint_data[Next_Wp][2]; //(Tempat Ketinggian)
      Sudut_ke_Tujuan = courseTo(loc1, loc2); //(Untuk sudut tujuan)
```

```
        PID_Roll();
        PID_Pitch();

        Jarak_ke_Tujuan = get_distance(loc1, loc2);
        if(Jarak_ke_Tujuan <= waypoint_radius){
            Next_Wp++;
            if(Next_Wp >= JUMLAH_WAYPOINT){
                    Next_Wp = 0;
            }
        }

}

void PID_Roll(){
        /*Lamanya perhitungan*/
        unsigned long now = timer2.get_micros()/1000;
        double timeChange_roll = (double)(now - lastTime_roll);

        Setpoint_roll = Sudut_ke_Tujuan;
         int heading_buf=(get_Heading()/5)*5;
        error_roll = Setpoint_roll - heading_buf;
        if (error_roll > 180) {
            error_roll = -1 * (360.00 - (Setpoint_roll - get_Heading()));
        }
        else if (error_roll < -180) {
            error_roll = (360.00 + (Setpoint_roll - get_Heading()));
        }

        errSum_roll = errSum_roll + (error_roll * timeChange_roll);
        double dErr_roll = (error_roll - last_error_roll) / timeChange_roll;

        /*Perhitungan keluaran PID*/
        PID_Output_roll = Kp_roll * error_roll + Ki_roll * errSum_roll +
        Kd_roll * dErr_roll; //(PID Roll)

        /*Sudut pergerakan sevo kendali pesawat*/
        last_error_roll = error_roll;
        lastTime_roll = timer2.get_micros()/1000;
```

```
    Roll_Servo = 1500 - PID_Output_roll;
    if(Roll_Servo >= 1800)      Roll_Servo = 1800; //(45 derajat
    servo = UP agar sevo tidak over maka 2000-200)
    if(Roll_Servo <= 1200) Roll_Servo = 1200;  //(45 derajat servo =
    DOWN servo tidak over maka 1000+200)
    set_Servo_Roll_Value(Roll_Servo);
}

void PID_Pitch(){ //(PID mengatur ketinggian)
    unsigned long now = timer2.get_micros()/1000;
    double timeChange_pitch = (double)(now - lastTime_pitch);

    Setpoint_pitch = waypoint_data[Next_Wp][2]; //(set point
    ketinggian)
    error_pitch = Setpoint_pitch - get_Altitude();

    errSum_pitch = errSum_pitch + (error_pitch * timeChange_pitch);
    double dErr_pitch = (error_pitch - last_error_pitch) /
    timeChange_pitch;

    /*PID keluaran untuk pitch ketinggian*/
    PID_Output_pitch = Kp_pitch * error_pitch + Ki_pitch *
    errSum_pitch + Kd_pitch * dErr_pitch;

    /* Sudut pergerakan sevo kendali pesawat */
    last_error_pitch = error_pitch;
    lastTime_pitch = timer2.get_micros()/1000;

    Pitch_Servo = 1500 - PID_Output_pitch;
    if(Pitch_Servo >= 1800)      Pitch_Servo = 1800; //(45 derajat
    servo = UP agar sevo tidak over maka 2000-200)
    if(Pitch_Servo <= 1200) Pitch_Servo = 1200; //(45 derajat servo
    = DOWN agar servo tidak over maka 1000+200)
    set_Servo_Pitch_Value(Pitch_Servo);
}

void Loiter_Mode(){

}
```

```
double get_PID_Output(void){
      return get_Servo_Roll_Value();
}

int get_error_pitch(){
  return error_pitch;
}
int get_error_roll(){
  return error_roll;
}

double get_Jarak_ke_Tujuan(){
  return Jarak_ke_Tujuan;
}
```

----------Myalgorithm.h----------

```
#ifndef MYALGORITHM_H_
#define MYALGORITHM_H_
#include <Arduino.h>
#include <eRCaGuy_Timer2_Counter.h>
#include "GPS.h"
#include "mySensor.h"
#include "myServo.h"

#define MANUAL_MODE          0
#define AUTO_MODE     1

void set_MODE(uint8_t mode);
uint8_t get_MODE(void);

void delay_T2(unsigned long time);

void init_waypoint(void);
void  set_waypoint_data(uint8_t  waypoint_number,  double  lat,  double
      lng);

void update_waypoint(void);
void PID_Roll();
void PID_Pitch();
```

```
void Loiter_Mode();
double get_PID_Output(void);
int get_error_pitch();
int get_error_roll();
double get_Jarak_ke_Tujuan();

#endif /* MYALGORITHM_H_ */
```

A.6 Listing Program Arduino pada mysensor.cpp dan mysensor.h

```
---------------------------------mysensor.cpp-----------------------------------
-
#include "mySensor.h"
MUTEX_DECL(dataMutex);

HMC5883L compass;
MPU6050 mpu;
#define P0 1013.25
BMP280 bmp;
// (Memasukkan sensor ketinggian)
SFE_BMP180 pressure;
double baseline; // (baseline pressure)
double baseline_temp; // (baseline pressure)

float heading1;
float heading2;

volatile double ketinggian;
volatile double heading_compass;

volatile double Ultrasonic_Distance;

int offset=0;

void init_MPU6050(void){
      // Initialize MPU6050(Accelero dan Gyro sensor)
        while(!mpu.begin(MPU6050_SCALE_2000DPS,
      MPU6050_RANGE_2G))
        {
          delay_T2(500);
```

```
        }

       // menggunakan sensor langsung
        mpu.setI2CMasterModeEnabled(false);
        mpu.setI2CBypassEnabled(true) ;
        mpu.setSleepEnabled(false);
}

void init_HMC5883L(void){
// Initialize Initialize HMC5883L(Kompas)
      while (!compass.begin())
      {
          delay_T2(500);
      }
// Set measurement range
      compass.setRange(HMC5883L_RANGE_1_3GA);
// Set measurement mode
      compass.setMeasurementMode(HMC5883L_CONTINOUS);
// Set data rate
      compass.setDataRate(HMC5883L_DATARATE_30HZ);
// Set number of samples averaged
      compass.setSamples(HMC5883L_SAMPLES_8);
// Set calibration offset. See HMC5883L_calibration.ino
      compass.setOffset(25, -197); //diganti 2 angka terakhir dari
      kalibrasi kompas dan acc
}

void init_BMP280(){
      if(!bmp.begin()){
         Serial.println("BMP init failed!");
         while(1);
        }
        else Serial.println("BMP init success!");

        bmp.setOversampling(4);

        baseline = 0;
        Serial.println("Calibrating BMP280...");
        for(int i=0;i<100;i++){
```

```
            getPressure_BMP280(baseline_temp);
            baseline += baseline_temp;
            delay_T2(20);
        }
        baseline = baseline/100;

        Serial.print("baseline pressure: ");
        Serial.print(baseline);
        Serial.println(" mb");
}
//(Untuk pembacaan ultrasonic)
void init_SRF02(void){
      Serial1.begin(9600);
}
//(Untuk pembacaan airspeed)
void init_MPXV7002dp(void){
   int sum=0;
   Serial.println("init Mpxv7002dp...");
   for(int i=0;i<10;i++)
   {
      sum += analogRead(A0)-512;
      delay_T2(10);
   }
   offset=sum/10.0;
}

float get_mpx_KMPH(void){
      float Vout = (5*(analogRead(A0)-offset))/1024.0;
      float P = Vout-2.5;
      return (sqrt(abs(2*P*3600/0.25))); //(merubah dari satian Pa
      menjadi Km/Jam)
}

void SendCmd(unsigned char address,unsigned char cmd){
      Serial1.write(address);//set the address of SRF02(factory default is
      0)
}

bool get_SRF02(void){
```

```
    SendCmd(0x00,0x54);          // (dalam cm dan otomatis kirim
    setelah selesai mengukur)
}

void set_UltraSonic_Distance(double mDistance){
    Ultrasonic_Distance = mDistance;
}

double get_UltraSonic_Distance(void){
    return Ultrasonic_Distance;
}
//( Penggunaan accelerometer dan kompas)
double calc_and_get_heading(){
    // (Pembacaan vektor)
        Vector mag = compass.readNormalize();
        Vector acc = mpu.readScaledAccel();

    // (Perhitungan arah)
        heading1 = noTiltCompensate(mag);
        heading2 = tiltCompensate(mag, acc);

        if (heading2 == -1000)
        {
                heading2 = heading1;
        }
        float declinationAngle = (1.0 + (1.0 / 60.0)) / (180 / M_PI);
        //(untuk wilayah surabaya)
        heading2 += declinationAngle;
        // Correct for heading < 0deg and heading > 360deg
        heading2 = correctAngle(heading2);
        heading2 = heading2 * 180/M_PI;
        return heading2;
}

// (Tidak memakai tilt compensation karena kondisi pesawat datar)
float noTiltCompensate(Vector mag)
{
 float heading = atan2(mag.YAxis, mag.XAxis);
 return heading;
```

```
}

// (Menggunakan tilt compensation pada pesawat kondisi miring)
float tiltCompensate(Vector mag, Vector normAccel)
{
  // Pitch & Roll

  float roll;
  float pitch;

  roll = safe_asin(normAccel.YAxis);
  pitch = safe_asin(-normAccel.XAxis);

  if (roll > 0.78 || roll < -0.78 || pitch > 0.78 || pitch < -0.78)
  {
    return -1000;
  }

// (Algoritma 2 kendali)
  float cosRoll = cos(roll);
  float sinRoll = sin(roll);
  float cosPitch = cos(pitch);
  float sinPitch = sin(pitch);

// Tilt compensation(Penggabungan antara accelerometer dan
     kompas)
  float Xh = mag.XAxis * cosPitch + mag.ZAxis * sinPitch;
  float Yh = mag.XAxis * sinRoll * sinPitch + mag.YAxis * cosRoll -
       mag.ZAxis * sinRoll * cosPitch;

  float heading = atan2(Yh, Xh);

  return heading;
}

// Correct angle
float correctAngle(float heading)
{
  if (heading < 0) { heading += 2 * PI; }
```

```
  if (heading > 2 * PI) { heading -= 2 * PI; }

  return heading;
}

bool getPressure_BMP280(double &tekanan)
{
  double T,P;
    char result = bmp.startMeasurment();

    if(result!=0){
    delay_T2(result);
      result = bmp.getTemperatureAndPressure(T,P);

        if(result!=0)
        {
          double A = bmp.altitude(P,P0);
          tekanan = P;
          return(true);
//          return(P);
        }
        else {
        return false;
        }
    }
    else {
        return false;
    }
}

double get_current_Altitude_BMP280(void){
        double a,P;
        // Get a new pressure reading:
          if(getPressure_BMP280(P)){
              // (Menampilkan perbedaan ketinggian relatif pada saat
        inisialisasi)
              a = bmp.altitude(P,baseline);
              return a;
          }
```

```
        else return -1;          //(error)
}

float safe_asin(float v)
{
   if (isnan(v)) {
      return 0.0f;
   }
   if (v >= 1.0f) {
      return PI/2;
   }
   if (v <= -1.0f) {
      return -PI/2;
   }
   return asinf(v);
}

void set_Heading(double mheading){
       heading_compass = mheading;
}

double get_Heading(void){
       return heading_compass;
}

void set_Altitude(double maltitude){
       // Lock access to data.
       chMtxLock(&dataMutex);
       ketinggian = maltitude;
       // Unlock data access.
       chMtxUnlock(&dataMutex);
}
double get_Altitude(void){
       double alt=0;
       // Lock access to data.
       chMtxLock(&dataMutex);
       alt = ketinggian;
       // Unlock data access.
       chMtxUnlock(&dataMutex);
```

```
    return alt;
}

----------------------------------------------mysensor.h-----------------------------
#ifndef MYSENSOR_H_
#define MYSENSOR_H_

#include <Arduino.h>
#include <ChibiOS_AVR.h>
#include <Wire.h>
#include <HMC5883L.h>
#include <MPU6050.h>
#include <SFE_BMP180.h>
#include <BMP280.h>
#include "myAlgorithm.h"

void init_MPU6050(void);
void init_HMC5883L(void);
void init_BMP280(void);
void init_SRF02(void);
void init_MPXV7002dp(void);

float get_mpx_KMPH(void);

void SendCmd(unsigned char address,unsigned char cmd);
bool get_SRF02(void);
double get_UltraSonic_Distance(void);
void set_UltraSonic_Distance(double mDistance);

double calc_and_get_heading(void);

float noTiltCompensate(Vector mag);
float tiltCompensate(Vector mag, Vector normAccel);
float correctAngle(float heading);

bool getPressure_BMP280(double &preassure);
double get_current_Altitude_BMP280(void);

float safe_asin(float v);
```

```
void set_Heading(double mheading);
double get_Heading(void);

void set_Altitude(double maltitude);
double get_Altitude(void);

#endif /* MYSENSOR_H_ */
```

A.7 Listing Program Arduino pada servo.cpp dan servo.h

```
----------------------------------------servo.cpp----------------------------------
-
#include "myServo.h"

#define Servo_Roll_PIN   5 //(Pin digunakan untuk Ch 1)
#define Servo_Pitch_PIN  6 //(Pin digunakan untuk Ch2)

Servo Servo_Roll;
Servo Servo_Pitch;

volatile int Servo_Roll_Value = 1500; //(Posisi aileron netral pada
      PWM 1500)
volatile int Servo_Pitch_Value = 1500; //(Posisi elevator netral pada
      PWM 1500)
void init_servo(void){
      Servo_Roll.attach(Servo_Roll_PIN);
      Servo_Pitch.attach(Servo_Pitch_PIN);
}

void set_Servo_Roll_us(int value){
      Servo_Roll.writeMicroseconds(value);
}
void set_Servo_Pitch_us(int value){
      Servo_Pitch.writeMicroseconds(value);
}

void set_Servo_Roll_Value(int value){
      Servo_Roll_Value = value;
}
```

```
void set_Servo_Pitch_Value(int value){
	Servo_Pitch_Value = value;
}
int get_Servo_Pitch_Value(void){
	return Servo_Pitch_Value;
}
int get_Servo_Roll_Value(void){
	return Servo_Roll_Value;
}
```

----------servo.h----------

```
#ifndef MYSERVO_H_
#define MYSERVO_H_
#include <Servo.h>
#include <Arduino.h>

void init_servo(void);
void set_Servo_Roll_us(int value);
void set_Servo_Pitch_us(int value);
void set_Servo_Roll_Value(int value);
void set_Servo_Pitch_Value(int value);
int get_Servo_Pitch_Value(void);
int get_Servo_Roll_Value(void);


#endif /* MYSERVO_H_ */
```

[Halaman ini sengaja dikosongkan]

RIWAYAT HIDUP

Erwan Aprilian, dilahirkan di Malang, 29 April 1984, merupakan putra pertama dari pasangan Bapak Herwanto dan Ibu Warlikah. Penulis menamatkan sekolah di SDN Tamanharjo 01 Singosari Malang tahun 1996. Kemudian masuk ke SLTPN 03 Singosari tamat tahun 1999, dan melanjutkan di SMAN 1 Lawang tahun 2002, pada tahun 2004, penulis melanjutkan pendidikan di Akademi Angkatan Udara (AAU) Majoring Elektronika dan tamat pada tahun 2007. Selanjutnya penulis mendapatkan tugas belajar S1 program Lintas Jalur di jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember Surabaya pada pertengahan tahun 2014. Penulis memilih bidang studi Elektronika dan mengambil topik Tugas Akhir di Laboraturium industri.

E-mail : erwanaprilian@gmail.com

[Halaman ini sengaja dikosongkan]